G E S : A DATA-STRUCTURE-TO-GPSS ENCODING SYSTEM

Richard Carl Hansen

United States
Naval Postgraduate School

THESIS

G E S : A DATA-STRUCTURE-TO-GPSS ENCODING SYSTEM

by

Richard Carl Hansen

December 1970

G E S : A Data-Structure-to-GPSS Encoding System

by

Richard Carl Hansen
Lieutenant Commander, United States Navy
B.A., University of California at Los Angeles, 1961

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the
NAVAL POSTGRADUATE SCHOOL
December 1970

ABSTRACT

A research effort currently underway at the Naval Postgraduate School deals with the design and implementation of a computer system for translating natural language descriptions of simulation problems into executable computer programs. In this system, English text is translated into an internal data structure, which can be considered to be the computer's "mental image" of a simulation problem. This internal data structure is then translated into a computer program which will perform the simulation.

This thesis reports on the design of an appropriate internal data structure for conveniently representing simple queueing problems and on the development of a procedure for translating such a data structure into a GPSS program. Also, pertinent aspects of the overall system are briefly described, and an example application to a specific problem is included.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

Since the dawn of the computer age, it has been the dream of many people to obtain a machine capable of comprehending and executing orders given in a natural language. Such a machine would undoubtedly be computer-oriented. In this report it will be referred to as a "natural language processor". A natural language processor should perform the following functions:

(1) Accept natural language statements and questions as input.

(2) Utilize syntactic and/or semantic procedures to translate the natural language into a formal processor language or internal data structure.

(3) Utilize deductive and/or inductive processes to perform the task(s) required (e.g. question-answer, mathematics word problem, simulation, etc.).

(4) Generate natural language strings as output.

There is a definite need for such a processor in the present-day environment. Many computer users are limited by modern computer languages to those tasks for which the language was specifically designed. In addition, the planning and writing of a computer program must be completed by the user prior to anything being accomplished by the computer itself. There are also many potential customers who do not utilize computers simply because they cannot or will not learn a language other than their own.

A. PRESENT-DAY TECHNOLOGY

A significant portion of modern computer research is devoted to the development of natural language processors. The major problem encountered thus far has been the accomplishment of function (2) above. That is to

say, it has become apparent that it is a formidable task to translate a natural language (e.g. English), with all its ambiguities and imprecision, into a precise and completely defined language or data structure with which the computer can accomplish its task(s). Most approaches to this problem have a basis in some proposed theory of language. The most widely accepted theory today is that of transformational grammar by Noam Chomsky [1]. Several other language theories are also available to the researcher, notably that of stratified grammar by Sydney Lamb [2, 3]. Several review articles have been written [4, 5, 6, 7, 8] containing information on some of the latest developments in the natural language processing field.

One natural language processor in its early stages of development is that of Professor George E. Heidorn at the Naval Postgraduate School in Monterey, California. His processor, hereafter referred to as NLP, is based upon Lamb's concept of a stratified grammar and, in addition, utilizes an entity-attribute-value internal data structure, hereafter referred to as IDS, for the formal representation of information. NLP is designed to be able to convert text strings of some input language (e.g. natural language, FORTRAN, user-defined, etc.) into an appropriate IDS and/or convert an IDS into text strings of an appropriate output language (e.g. natural language, user-defined, GPSS, etc.). Work currently being done with NLP is directed toward automating the task of converting a natural language simulation problem description into a simulation language computer program. Nothing has been published to date on this work, but a general description of those portions of NLP relevant to this thesis appears in Section II.

B. THESIS OBJECTIVE

The solutions to problems relating to natural language processor functions listed in (1), (2), and (4) above were considered to be beyond the scope of this thesis. Indeed, several computer scientists have devoted years of research to these functional areas and have produced very few useful results. The primary research objective of this thesis was therefore defined as: Given a description of a simple simulation problem in the IDS form produced by NLP, develop a process to convert the IDS information into a GPSS program. In addition, a secondary objective was defined as: Assist in developing the exact form of the IDS that would be most convenient for producing GPSS programs.

C. ORGANIZATION OF THESIS

The remainder of the thesis is divided into four sections. Section II is a general description of the relevant portions of NLP. Section III covers the development of the basic approaches to the achievement of the research objectives, together with the presentation of a system which meets those objectives. Section IV is an application of that system to a specific simulation problem. Section V presents conclusions and recommendations.

II. NLP DESCRIPTION

Prior to undertaking the conversion of an IDS to a GPSS program via computer, an understanding of NLP is required. This section is therefore devoted to a general description of the pertinent parts of NLP. Subjects associated with the decoding (i.e. conversion of text to an IDS) process are not discussed, as they are not relevant to the achievement of the thesis objectives.

A. CELL AND RECORD STRUCTURE

The basic building block of the IDS is called a "cell" (Figure 1). A cell, when utilizing an IBM-360/67, consists of sixty-four bit positions arbitrarily numbered 1 to 64 from right to left. There are four parts to most cells. Bits 64 through 49 are called the TYPE, bits 48 through 33 are called the ATTR (for attribute), bits 32 through 17 are called the ADDR (for address), and bits 16 through 1 are called the LINK. TYPE is a number which specifies the type (0, 1, 2, or 3) of the cell. ATTR is a number which specifies the attribute number of the cell. ADDR is a number which is the information content of the cell. It may stand by itself as a numeric value or may point to another cell. LINK is a pointer to the next cell in a "list". If LINK is zero, it signifies the end of the list.

A TYPE 0 cell (Figure 2a) carries its ultimate information in the ADDR part of the cell. This essentially means that a TYPE 0 cell is used to represent numeric information. A TYPE 1 cell (Figure 2b) uses ADDR to point to another cell. This other cell then contains either individual-bit-oriented information or the EBCDIC representation of eight characters. A TYPE 2 cell (Figure 2c) uses ADDR to point to a "local list" of other

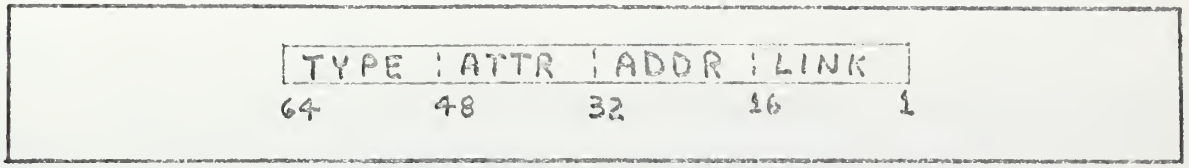


FIGURE 1 - GENERAL CELL STRUCTURE

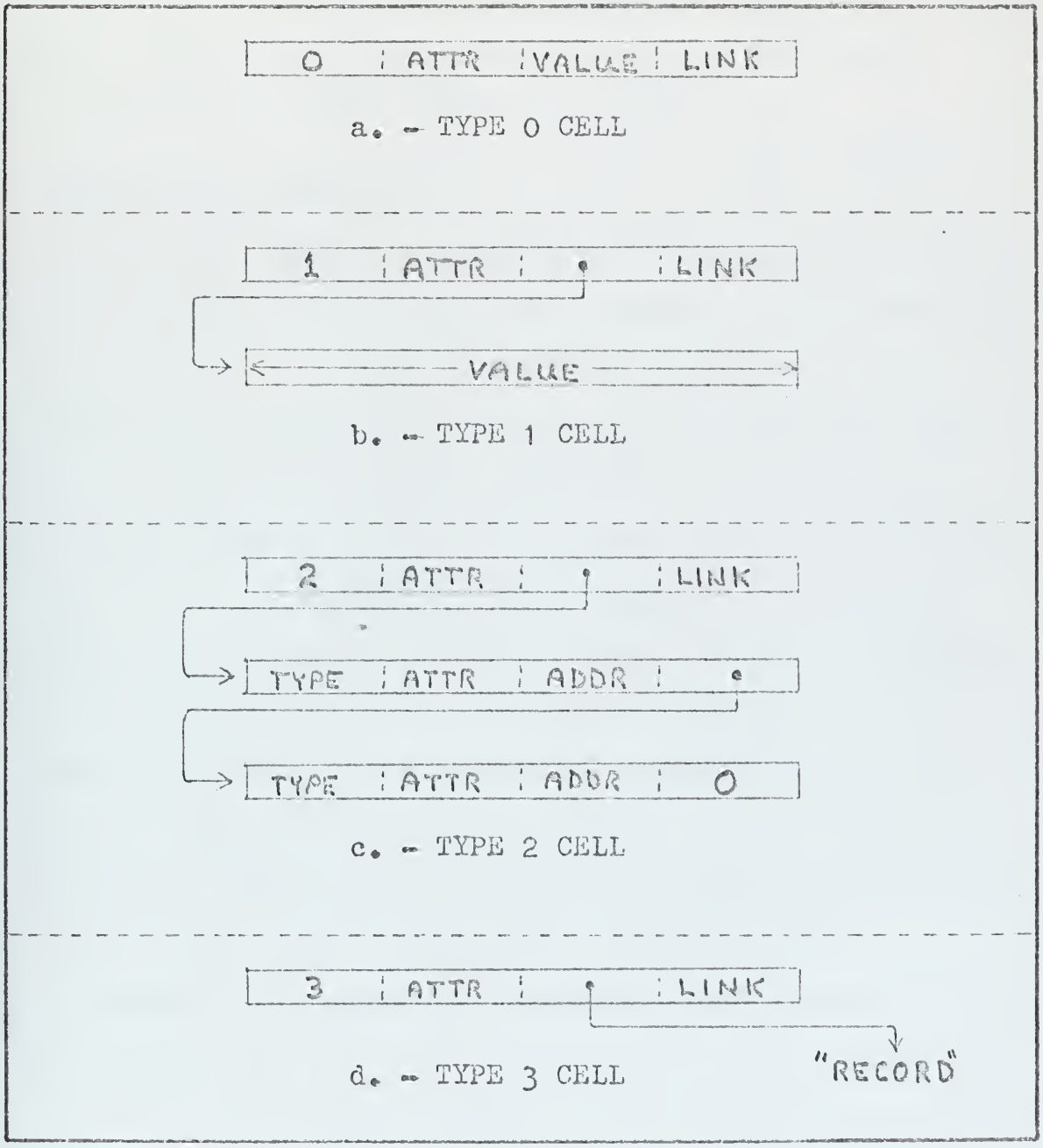


FIGURE 2

cells. The cells on this local list may be of any TYPE. A TYPE 3 cell (Figure 2d) uses ADDR to point to a "record".

A "list" is a group of cells, successively LINKed, in which the last cell of the list has a LINK of zero. A "local list" is any list other than a "record". A "record" (Figure 3) is a special kind of list in which the first cell on the list is of TYPE 0 and uses its ADDR as a "reference counter". The reference counter designates how many TYPE 3 cells, located elsewhere in the IDS, point to this record.

B. ATTRIBUTES AND INDICATORS

A record may be thought of as representing some unique item such as a book, ship, person, action, thought, word, sentence, etc. In other words, a record represents an "entity", and this entity possesses certain distinguishing "attributes", the values of which are specified in the record. For example, a ship is an entity and it has an attribute of entity type (attribute 1), the value of which is 'ship', and an attribute of 'color', the value of which is 'blue' (Figure 4). The attribute designation is specified in the ATTR of a cell. It may be an arbitrary number that has a unique meaning for that particular record (e.g. attribute 1 is synonymous with entity type attribute), or it may be a pointer to a "named record" (NAMREC). A NAMREC is a record which has a name attribute (ANMS) with a value which is the EBCDIC representation of the record's name. Figure 5 illustrates a NAMREC for 'ship'.

An indicator is a special type of attribute. When the value of an attribute can be specified by either 0 or 1, then only one bit position is required for the value. An example of this situation would be a record which represents a verb phrase in English. One attribute of the verb phrase is whether or not it is negative (e.g. "is <u>not</u> watching"). The

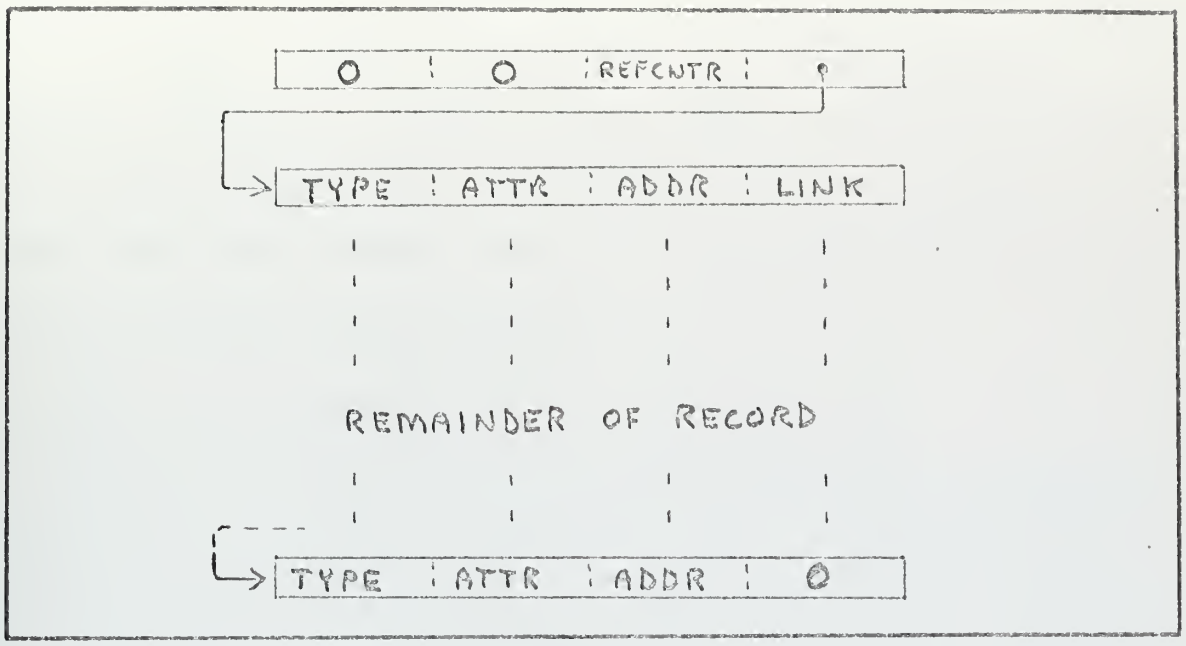


FIGURE 3 – GENERAL RECORD STRUCTURE

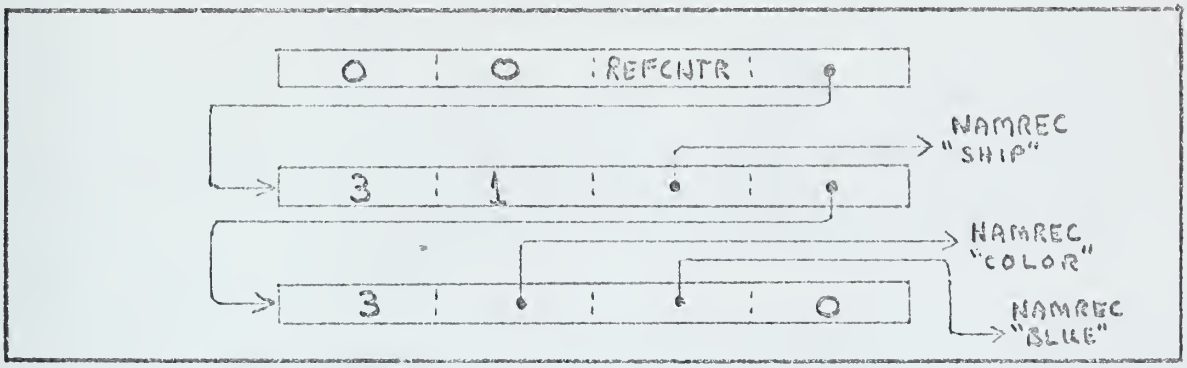


FIGURE 4 – RECORD WITH ATTRIBUTES

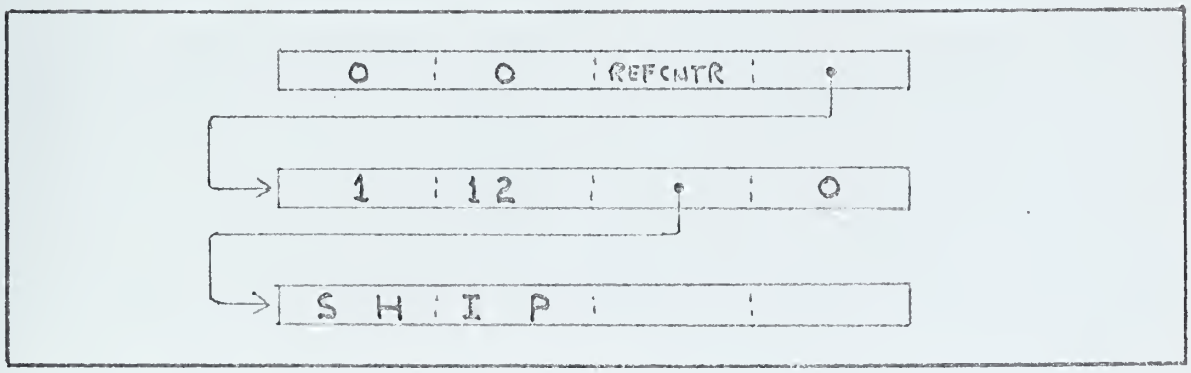


FIGURE 5 – NAMED RECORD FOR "SHIP"

negative attribute can be specified yes or no, 0 or 1. To take advantage of this feature, one attribute used by NLP, called INDICATORS and designated attribute 2, utilizes the individual bit positions in a cell for a large number of on/off value attributes. Figure 6 lists some of these on/off value attributes used in NLP processing of IDS into English text and illustrates an example IDS.

C. SEGMENTS AND SEGMENT TYPES

A SEGMENT is a special type of record that represents the information in a portion of the IDS being processed by NLP. A SEGMENT may represent one singular piece of information or it may represent many. A SEGMENT possesses varying numbers of attributes depending upon its use, that is to say, the number of attributes increases with an increase in the complexity of the information. For example, the SEGMENT representing "the" may have only two attributes, while the SEGMENT representing "The ship sailed into port." may have fifteen or twenty. Just as a portion of the IDS may represent an entity, so too may a SEGMENT represent that entity during the processing.

A SEGMENT may be any one of a number of different types. The type of a SEGMENT depends upon its use. For example, the SEGMENT representing the character "m" would be of a type known as a terminal SEGMENT (i.e. one which represents single printable characters). On the other hand, a SEGMENT representing the verb phrase "had been watched" would be of a type known as a verb-phrase SEGMENT. The record containing the name of the type of a SEGMENT is called the SEGMENT TYPE. A SEGMENT TYPE (note that SEGMENT TYPE is different from SEGMENT type) is a record possessing an ANMS attribute with the name of the SEGMENT type as its value. The SEGMENT TYPE and the SEGMENT itself are usually referred to by this ANMS

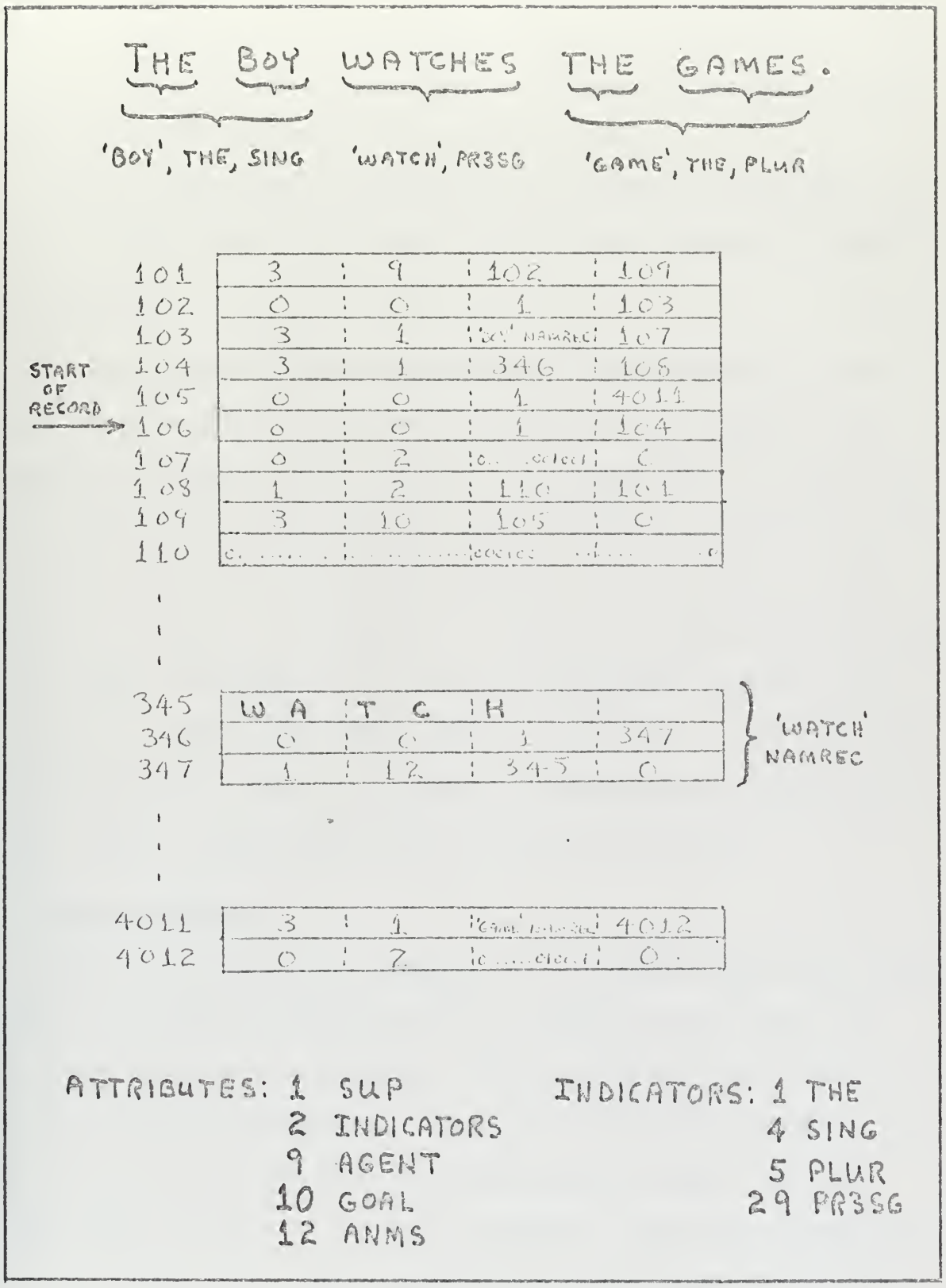


FIGURE 6 - ENGLISH TEXT IDS

value. The two examples above would therefore be referred to as M and VERBPH. M would require no real explanation as to exactly what it represented, but VERBPH would possess several attributes which would precisely define its content and location in a body of text.

Also, a SEGMENT TYPE record possesses an attribute which points to a list of encoding "rules" that begin with that particular type of SEGMENT.

D. NLP RULES

NLP uses a group of processing statements called "rules" for the conversion of an IDS to strings of a specified output language. This conversion process is called "encoding". The application of a rule consists of first testing the condition of the SEGMENT, as specified on the left side of the rule, to see that it possesses the desired attribute values to make the rule applicable. If the rule is found to be applicable, a new SEGMENT (or SEGMENTS) is constructed with unique attribute values, as specified on the right side of the rule. Appendix A presents the BNF description for NLP rules and Appendix B explains NLP rule symbology and usage. Figure 7 illustrates several example encoding rules.

E. ENCODING PROCEDURE

Prior to the start of encoding any IDS, the SEGMENT TYPE pointer (STP) points to a starting SEGMENT TYPE and the SEGMENT pointer (SP) is null (i.e. does not point to any record). Both the STP and SP are placed on the top of a two-column push-down stack. When the rule scan begins, the STP and SP on top of the stack are removed from the stack. Then the SEGMENT TYPE pointed to by the STP is checked to acquire the appropriate list of encoding rules. The applicable rule in the list is determined by comparison of the attribute conditions located to the left of the conversion symbol (-->) with conditions prevailing in the SEGMENT pointed to

```
SENT --> FINCL (% SENT) .
FINCL --> VERBPH (% FINCL)
VERBPH (¬ SUBJ, ¬ AGENT, ¬ PASSIVE) --> ...
        VERBPH (PASSIVE)
VERBPH ('BE', FINITE, NOT) --> VERBPH (-NOT) ...
        ADV ('NOT')
NOUNPH (INTERR) --> ADJPH ('WHICH') NOUNPH
        (-INTERR, -THEAAN, -OWNER)
VERBP ('BE', PR3SG) --> I S
VERBP ('GIV', PAST) --> G A V E
ADJP (THE) --> T H E
```

FIGURE 7 - NLP ENCODING RULE EXAMPLES

by the SP. When the applicable rule is found (or the non-discovery default option is taken), new SEGMENTS are constructed according to the labeling specifications located to the right of the conversion symbol. These new SEGMENTS each have their own STP and SP. The old STP and SP are erased as well as the SEGMENT pointed to by the old SP. The new STP's and SP's are then placed on the stack in reverse order (i.e the STP and SP for the SEGMENT constructed nearest the conversion symbol are placed on the stack last).

It should be noted that the erasure procedure does not result in the destruction of any original record structure (IDS). That is, the first SEGMENT type to be processed in any IDS always results in new SEGMENTS being constructed. Subsequent rule processing is performed on copies of those SEGMENTS or on other copies of original records. The only changes that are made to original record structure are accomplished by adding, deleting, or modifying attributes via the record MEMORY, which points to original records.

III. GPSS ENCODING SYSTEM DEVELOPMENT

In accordance with the research objectives set forth in the INTRODUCTION, a GPSS Encoding System (GES) was developed. The scope of the task included both specifying the appropriate form of the IDS and developing the procedures necessary for conversion of the IDS into GPSS text. An initial problem was determining the type of system the GES was to be. The remainder of this section is devoted to a short discussion of some of the general problems encountered, a description of both the IDS form and conversion procedures developed, and the manual processing of a sample queuing problem to illustrate the application of those procedures.

A. GENERAL PROBLEMS

It was decided at the beginning of the research to concentrate most of the effort toward developing general purpose sections of GPSS code for simple queuing situations. Then, should time permit, the situations could be increased in complexity and the GES expanded to accommodate them. In any event, certain fundamentals of GPSS programming (e.g. time, queues, storages, distributions, etc.) were to be developed in the GES for the most general application possible.

The primary problem encountered during the early stages of GES development was that both the form of the IDS and the procedures for IDS conversion to GPSS program had to be developed simultaneously. Formal documentation of GPSS programming existed [9, 10] and the principles concerning IDS building blocks, as presented in Sections II.A. and II.B., were fairly well set. Other than those restrictions, however, no rigid guidelines were specified for GES development.

A system-question/user-answer scheme was studied for possible applications. This approach would have been generally modeled after a programming-by-questionnaire study of job-shop simulation conducted at an earlier date by the RAND Corporation [11]. Although the scheme was soon rejected, the idea of a system being able to detect and request missing information necessary for the simulation was retained for future development should time permit.

The NLP concepts of the IDS were utilized in the GES. The IDS was constructed to approximate the mental image an analyst may maintain of a simulation problem. Since a record may represent an entity of almost any description, an IDS constructed of linked records could easily be used for GPSS-oriented requirements.

The scheme of using the NLP rule form and symbology for the conversion process was adopted. The main reason behind this decision was that the NLP rules were already a workable process for the conversion of IDS to English text and that the simplest procedure to follow would be to use the same idea for conversion of IDS to GPSS text. By the adoption of this scheme, the time-consuming development of alternate conversion process programming was avoided. However, the capabilities of the NLP rule form needed to be expanded. This expansion was accomplished by Professor Heidorn concurrently with GES development.

B. INTERNAL DATA STRUCTURE FORM

The basic reasoning underlying the particular IDS form chosen for the GES was that the primary units in any simulation problem are ENTITIES (e.g. man, ship, gas station, store, etc.), ACTIONS (e.g. arrive, go, buy, unload, etc.), ATTRIBUTES of those ENTITIES and ACTIONS (e.g. quantity, location, color, agent, goal, etc.), and VALUES of the ATTRIBUTES (e.g.

5, downtown, red, man, cargo, etc.). These primary units were combined in an overall scheme of problem representation that places the greatest importance on records which represent ACTIONS. This scheme specifies that a transaction (ENTITY) proceeds chronologically from ACTION to ACTION, each of which takes place at some location (ENTITY). For example, a 'ship' (ENTITY) proceeds to an 'unload'(ACTION) which takes place at a 'dock' (ENTITY). The location ATTRIBUTE in the example is assigned to the ACTION, but, if the dock were located in a harbor, then the ENTITY 'dock' would have a location ATTRIBUTE with a value of the ENTITY 'harbor'. All records used in the IDS of the GES have ATTRIBUTES, the value of any one of which may be:

(1) A pointer to an ACTION record.

(2) A pointer to an ENTITY record.

(3) A pointer to an ATTRIBUTE Descriptor record.

(4) A numerical value (e.g. 53, 812, etc.)

(5) A pointer to a non-numerical value (e.g. 'red', 'heavy', etc.)

Appendices C and D define the various types of records and associated ATTRIBUTES used in GES. Appendix E lists the named records developed for GES usage.

A sample simulation problem fact summary is shown in Figure 8, while Figure 9 presents a simplified graphic representation of the problem. Figure 10 illustrates the IDS cell content for a portion of the problem. Figure 11 shows the IDS for the sample problem in a "direct-specification" format. This format is identical to the one used by NLP to input an explicit specification of the IDS.

Black ships arrive at a dock. The dock has a capacity of one unit and each black ship takes up one unit of capacity. The interarrival time is 2/3 hour. After arrival at the dock, a black ship unloads cargo. Unloading time is 30 minutes, exponentially distributed. After unloading, a black ship leaves the dock. The basic time unit is the minute. Problem time is 8 hours.

FIGURE 8 - SAMPLE PROBLEM FACT SUMMARY

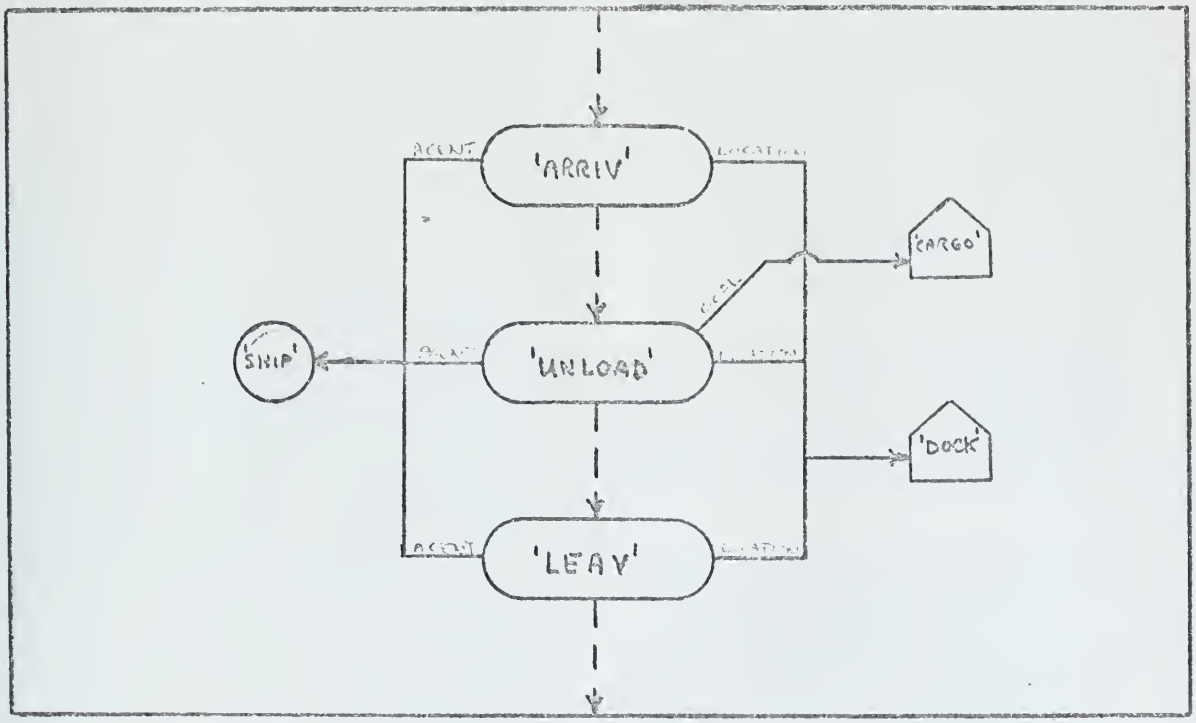


FIGURE 9 - SAMPLE PROBLEM IDS GRAPHIC REPRESENTATION

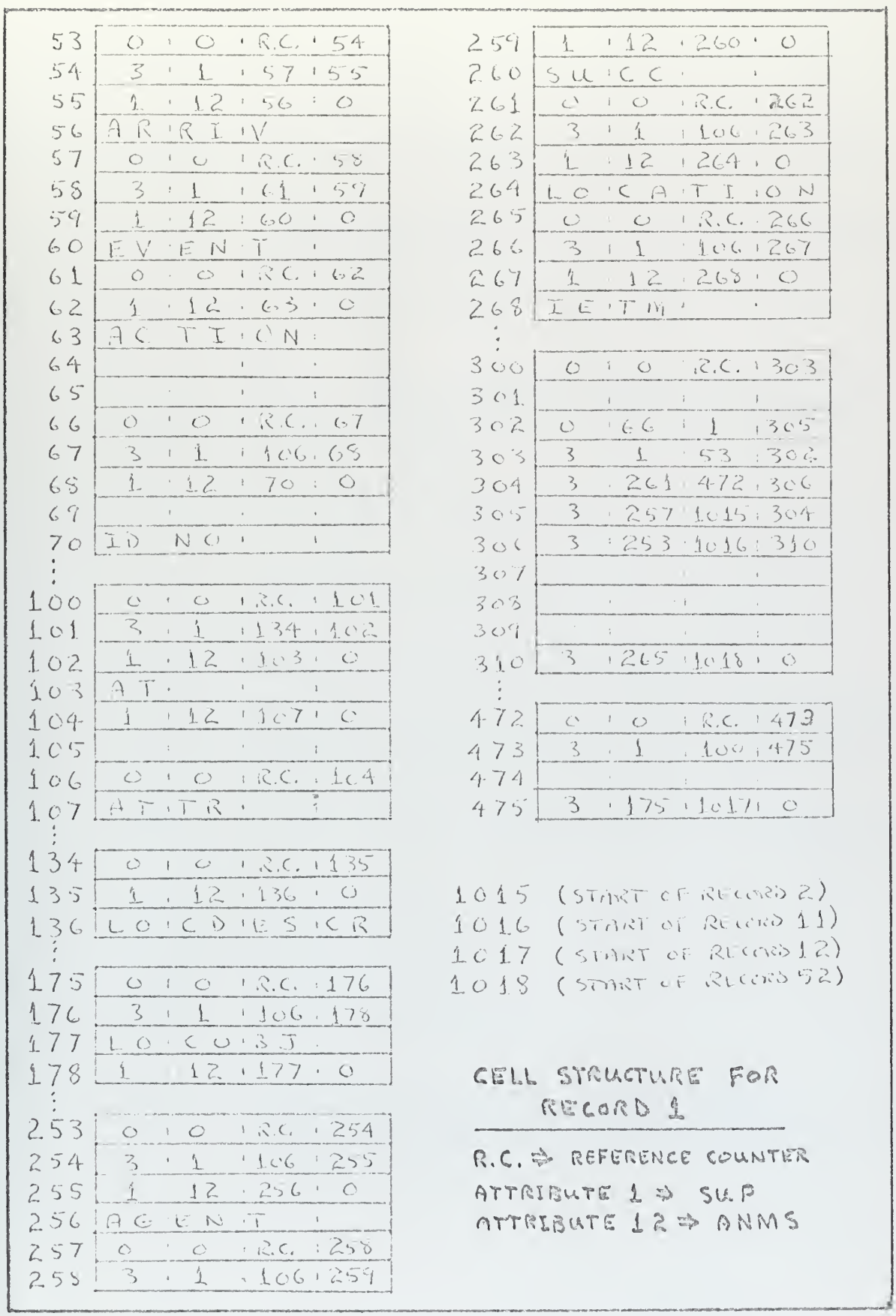


FIGURE 10 – SAMPLE PROBLEM CELL STRUCTURE EXAMPLE

```
'REC1'    ('ARRIV',IDNO=1,SUCC='REC2',LOCATION='REC21',
           AGENT='REC11',JETM='REC52')

'REC2'    ('UNLOAD',IDNO=2,PRED='REC1',SUCC='REC3',
           LOCATION='REC21',AGENT='REC11',GOAL='REC13',
           DURATION='REC31')

'REC3'    ('LEAV',IDNO=3,PRED='REC2',AGENT='REC11',
           LOCATION='REC21')


'REC11'   ('SHIP',IDNO=1,CONSUMP='REC51',COLOR='BLACK')

'REC12'   ('DOCK',IDNO=2,QUANTITY=1,CAPACITY='REC51')

'REC13'   ('CARGO',IDNO=3)


'REC21'   ('AT',LOCOBJ='REC12')


'REC31'   ('EXPON',MEAN='REC53')


'REC41'   ('MOBLIST',LASTREC=11,@11='REC11')

'REC42'   ('STALIST',LASTREC=12,@11='REC12',@12='REC13')

'REC43'   ('ACTNLIST',LASTREC=13,@11='REC1',@12='REC2',
           @13='REC3')

'REC44'   ('DSTRLIST',LASTREC=11,@11='REC31')

'REC45'   ('SCSRLIST',LASTREC=12,@11='REC2',@12='REC3')


'REC51'   ('UNIT',NUM=1)

'REC52'   ('MINUTE',NUM=40)

'REC53'   ('MINUTE',NUM=30)

'REC54'   ('MINUTE',NUM=480)


SETMEM    (RNNO(MEMORY)=1,MEPTR(MEMORY)='REC41',
           SEPTR(MEMORY)='REC42',ACPTR(MEMORY)='REC43',
           DISTPTR(MEMORY)='REC44',SUCPTR(MEMORY)='REC45',
           PROBTIME(MEMORY)='REC54',MFNID(MEMORY)=3,
           MVARID(MEMORY)=1)
```

FIGURE 11 - SAMPLE PROBLEM IDS

C. GES RULES

Appendix F defines various segment types used in GES and lists their normally-held attributes or usual record type. Appendix G lists the GES rules developed to date. Each rule is numbered for reference. Before a detailed explanation of some of the more illustrative rules can be given, the general processing of a problem IDS will be reviewed in terms of overall GES rule logic.

1. General Processing Logic

At the start of IDS processing, the record MEMORY is examined to locate a list (SELIST) of all stationary (i.e. non-self-propelled) entities (STENTITY's) in the problem. Each STENTITY is examined to determine its eligibility as a GPSS storage. If applicable, a storage definition block is printed using the IDNO attribute of the STENTITY as storage identification. After the last STENTITY is processed, preset information is printed which consists of an RMULT block, an exponential distribution function definition (PRESET1), and a normal distribution function definition (PRESET2). Processing continues with the location and examination of a list of distributions (DISTLIST). Each distribution requiring a function definition (FNDEF) has one printed using the FNID attribute as function identification. Next, a list of successors (SUCLIST) is examined for those successors requiring a function definition (SUCREC). Each SUCREC requiring a definition has one printed using the MFNID attribute of MEMORY as a sequential identification number. Finally, a list of actions (ACLIST) is located which contains each separable action (ACT) in the problem. At this point, the processing reaches the heart of the program (i.e. the production of executable GPSS blocks). Each ACT is examined for particular attribute conditions, the

satisfaction of which will result in a group of BLOK-type segments. These groups of segments will become the general purpose sections of GPSS code mentioned in Section II.A.

A group of BLOK-type segments represents either a first, last, or an intermediate action in the life of an applicable transaction. The first action is usually some sort of arrival to the problem area and the last action is usually some sort of leaving from that same area. An intermediate action consists of waiting in a local queue for admittance to either a facility or storage. Once inside the facility or storage, time advances or the transaction awaits the fulfillment of some condition. Then the transaction leaves the facility or storage and proceeds to the next appropriate action as designated by a successor description. All actions except last actions have some sort of successor description. It may simply be a direction to the next action on the list, in which case no GPSS output is required. On the other hand, the next action to which the transaction must proceed may depend upon a number of different items, including the type of transaction involved or an arbitrary percentage assignment. In any event, multi-path transaction processing is accomplished through the use of the successor description.

Each BLOK-type segment is processed for possible block labeling or identification numbering, for the name of the block (BLOKNAM), and for arguments (BLOK1). Arguments usually require further processing until printout is reached. It should be noted that although the BLOK,BLOKNAM, BLOK1, etc. processing was explained only for ACT's, it is applicable to almost all higher level segments in GES.

One of the more interesting features involved in GES rule logic is that the identification number of a stationary entity is used to

identify the queue, facility, or storage associated with it as a location. Also, for transaction routing within the GPSS program, a block label is designated for the first block in a group. The label consists of the letters "LBL" followed by the identification number of the action which caused the group to exist. Lastly, an FVARIABLE definition block is produced whenever a normal distribution is used. The different FVARIABLE's in a program are identified sequentially by the MVARID attribute of MEMORY.

2. Example GES Rule Explanations

This sub-section is devoted to detailed explanations of the GES process involving certain specific rules. The individual rules have been selected to provide a wide range of rule syntax and symbology usage, and are not particularly significant with regard to overall GES logic.

a. Rule Number 2

```
SELIST(LISTCNTR(MEMORY).LT.LASTREC)   -->   ...
    STENTITY(%@LISTCNTR(MEMORY)(SELIST))     ...
    SELIST(LISTCNTR(MEMORY)=LISTCNTR(MEMORY)+1)
```

The segment type pointer (STP) points to SELIST, or rather to a segment type with an ANMS attribute of "SELIST". The value of the attribute LISTCNTR of the record MEMORY is tested to see if it is less than the value of the attribute LASTREC of the segment pointed to by the segment pointer (SP), which is called the old SELIST. If this condition is met, two new segments are constructed. The first segment has an STP pointing to STENTITY and is a copy of the record pointed to by attribute number XX of the old SELIST, where XX is the value of attribute LISTCNTR of the record MEMORY. The second segment has an STP pointing to SELIST

and is a copy (by default) of the old SELIST. In addition, the attribute LISTCNTR of MEMORY is incremented by 1.

b. Rule Number 15

```
SUCREC($'SUCDSCR1')   -->   BLOK('FUNCTION',IDNO
    =MFNID(MEMORY),MFNID(MEMORY)=MFNID(MEMORY)
    +1,ARGA=SUCARG(SUCREC),TEMP=XYLAST(SUCREC)
    -100,ARGB=TEMP/2,DORC="D",TEMP(MEMORY)=101)  ...
    NEWLINE1   FNDEF1(%SUCREC)
```

The STP points to SUCREC. The SUP attribute of the old SUCREC is checked through succeeding records pointed to by SUP (called "checking the SUP chain") for the value 'SUCDSCR1'. Actually, the ANMS attribute of succeeding SUP records is checked for the value "SUCDSCR1". If that condition is satisfied, then three new segments are constructed. The first segment has an STP pointing to BLOK, a SUP attribute pointing to 'FUNCTION', an IDNO attribute whose value is the same as the attribute MFNID of MEMORY (then MFNID is incremented by 1), an ARGA attribute whose value is the same as the value of the attribute SUCARG of the old SUCREC, a TEMP attribute with a value of the quantity remaining when 100 is subtracted from the value of attribute XYLAST of the old SUCREC, an ARGB attribute with a value of the TEMP attribute value divided by 2, and a DORC attribute with a value of "D". The attribute TEMP of MEMORY is set to a value of 101. The second segment has an STP pointing to NEWLINE1. The third segment has an STP pointing to FNDEF1 and is a copy of the old SUCREC.

c. Rule Number 35

```
BLOCK(¬STORIND(ARGA),'ENTER'|'LEAVE')   -->   NULL
```

The STP points to BLOK. The record pointed to by the attribute ARGA of the old BLOK is checked to determine that it does not have a STORIND attribute. If this condition is satisfied, the SUP attribute of the old BLOK is checked to determine if it points to either 'ENTER' or 'LEAVE'. If this condition is met, a new segment is constructed. The new segment has an STP pointing to NULL.

d. Rule Number 54

ARGS('EXPON') --> ARG(%MEAN(ARGS)) , F N 1

The STP points to ARGS. The SUP attribute of the old ARGS is checked to see if it points to 'EXPON'. If the condition is met, five new segments are constructed. The first segment has an STP pointing to ARG and is a copy of the record pointed to by the MEAN attribute of the old ARGS. The other four segments are terminal segments and will eventually result in the printing of the characters represented (i.e. ,FN1).

e. Rule Number 62

ARG('PARAMNO') --> P NUMBER(NUM(ARG))

The STP points to ARG. The SUP attribute of the old ARG is checked to see if it points to 'PARAMNO'. If the condition is met, two new segments are constructed. The first is a terminal segment representing the character "P". The second segment has an STP pointing to NUMBER and a NUM attribute that is a copy of the NUM attribute of the old ARG.

f. Rule Number 72

NEWLINE1 --> OUTPUT(@11=1,@12=1)

The STP points to NEWLINE1. The condition is met by default and a new segment is constructed. The STP of the new segment points to OUTPUT, attribute number 11 is set to the value 1, and attribute number 12 is set to the value 1. When the OUTPUT segment is processed, it will result in the printer carriage control shifting to a new line and to column 1.

D. SAMPLE PROBLEM PROCESSING

Before proceeding with the description of GES processing applied to the sample problem shown in Figures 9 through 11, it is necessary to define some graphic notation which will simplify the explanation of the processing. The stack can be visualized as containing the segment types and segments themselves, rather than their pointers. The processing involving one applicable rule will be called a "step". Figure 12 is a graphical representation of a step. It shows the stack prior to the start of a rule scan with consequent removal of the top STP and SP; the processing via applicable rule (X), the stack after processing, and the resultant printout and/or column pointer (↑) placement until a new line is reached.

At the beginning of GES processing, the STP points to GPSSPROG and the SP is null. After processing is initiated, the top STP on the stack is removed and its segment type is checked for the list of appropriate rules. The rules on the list are then examined one-by-one in the order given on the list. The first (and only) rule on the GPSSPROG list has no conditions and is therefore automatically satisfied by default. This results in the construction of two new segments. One segment has an STP that points to BLOK and a SUP attribute that points to 'SIMULATE'. The other segment has an STP pointing to SELIST and is a copy of the record pointed to by attribute SEPTR of MEMORY. In addition, the attribute LISTCNTR of MEMORY is set to 11. This initial step is shown in graphic notation in Figure 13a.

The next processing step is not as simple as the first one. After the start of the scan of the BLOK rule list, the conditions of rule 35 are not met. Although the condition that ARGA does not have a STORIND

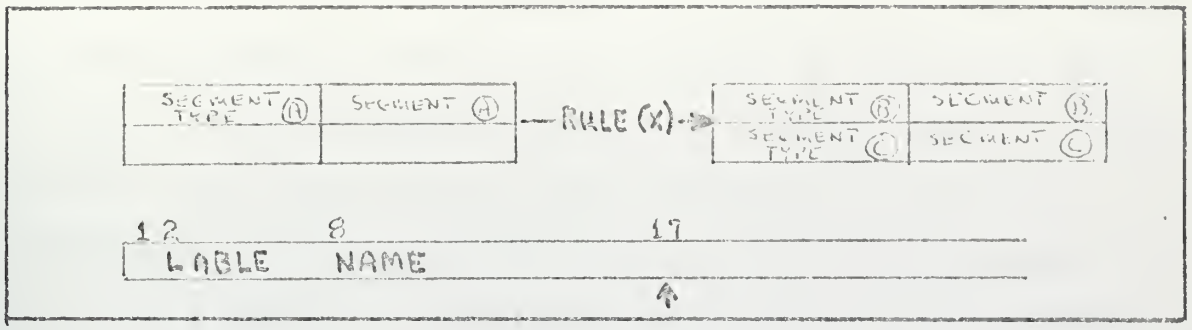
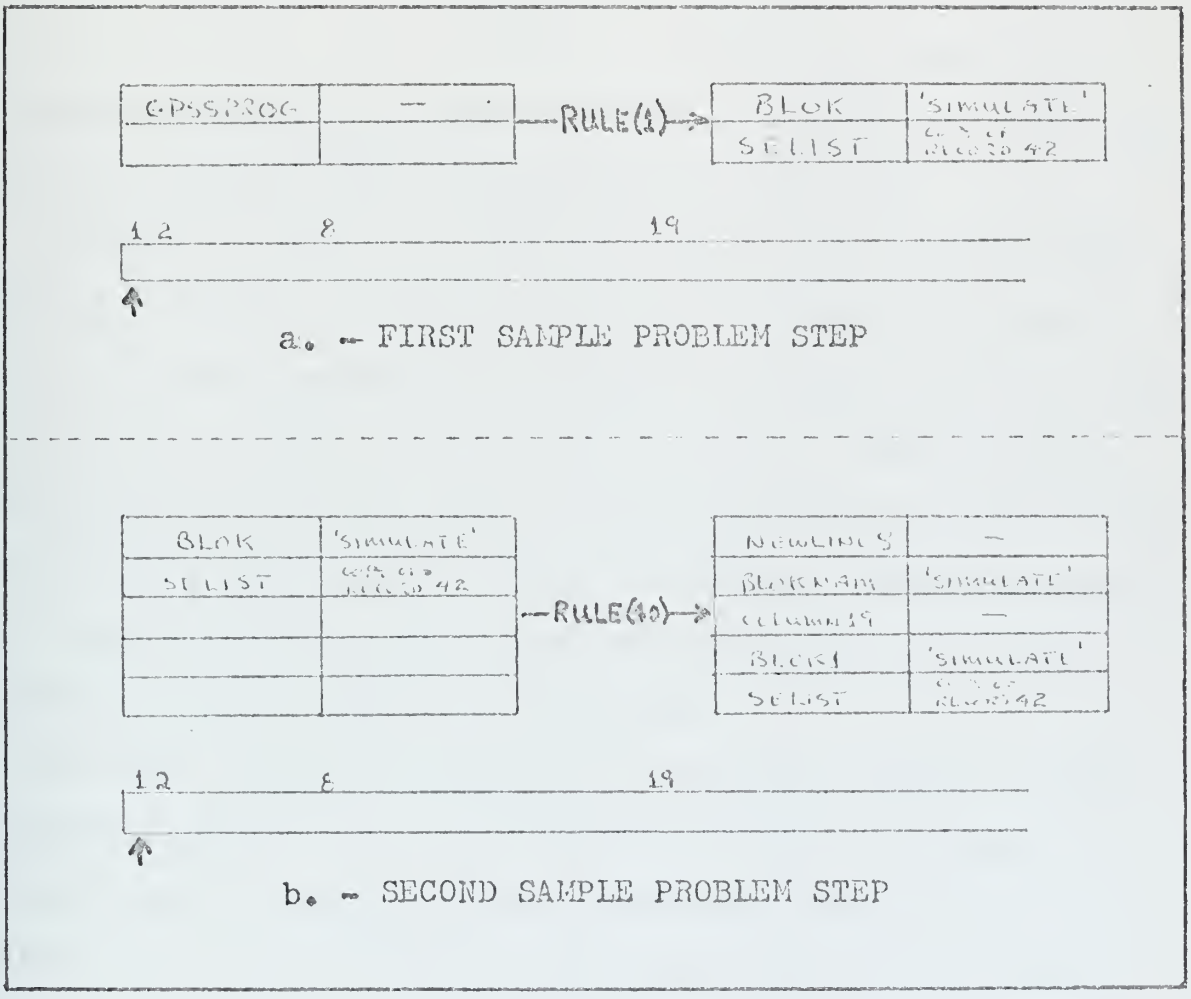


FIGURE 12 - GENERAL FORMAT FOR A STEP



a. - FIRST SAMPLE PROBLEM STEP

b. - SECOND SAMPLE PROBLEM STEP

FIGURE 13

attribute (actually the record pointed to by ARCA does not have a STORIND attribute) is satisfied (since the old BLOK does not even have an ARGA attribute), the SUP attribute does not point to either 'ENTER' or 'LEAVE'. Similarly, the old BLOK does not have a SUP attribute pointing to 'ADVANCE' and therefore the conditions of rule 36 are not satisfied. Also, the old BLOK has no LABL attribute, so rule 37 is not satisfied. Neither does the SUP attribute of the old BLOK point to 'TRANSFER' nor does the old BLOK possess an IDNO attribute, resulting in non-satisfaction of rules 38 and 39. Rule 40, however, is satisfied (by default) and processing results in the construction of four new segments. This second processing step is graphically represented in Figure 13b.

Inspection of Figures 13a and 13b reveals that the resultant stack from the first step is identical to the starting stack of the second step. The graphic notation will therefore be modified to the effect that the starting stack will be eliminated. In addition, "copy of record" will be represented by the symbol "%" and modifications of a copy will be carried in the same stack space as the % designation. Additional sequential processing of the sample problem IDS is presented in this newly modified notation in Figures 14a through 14k.

To further increase the efficiency of the graphic notation, a final modification will be made. The stack will no longer be illustrated, the column indicator will not be shown, and processed rules shall be listed by number only, grouped to provide one or more lines of printed output. The remainder of the sample problem IDS processing is shown in Figures 15a through 15n. The entire printed output for the sample problem processing is consolidated in Figure 16.

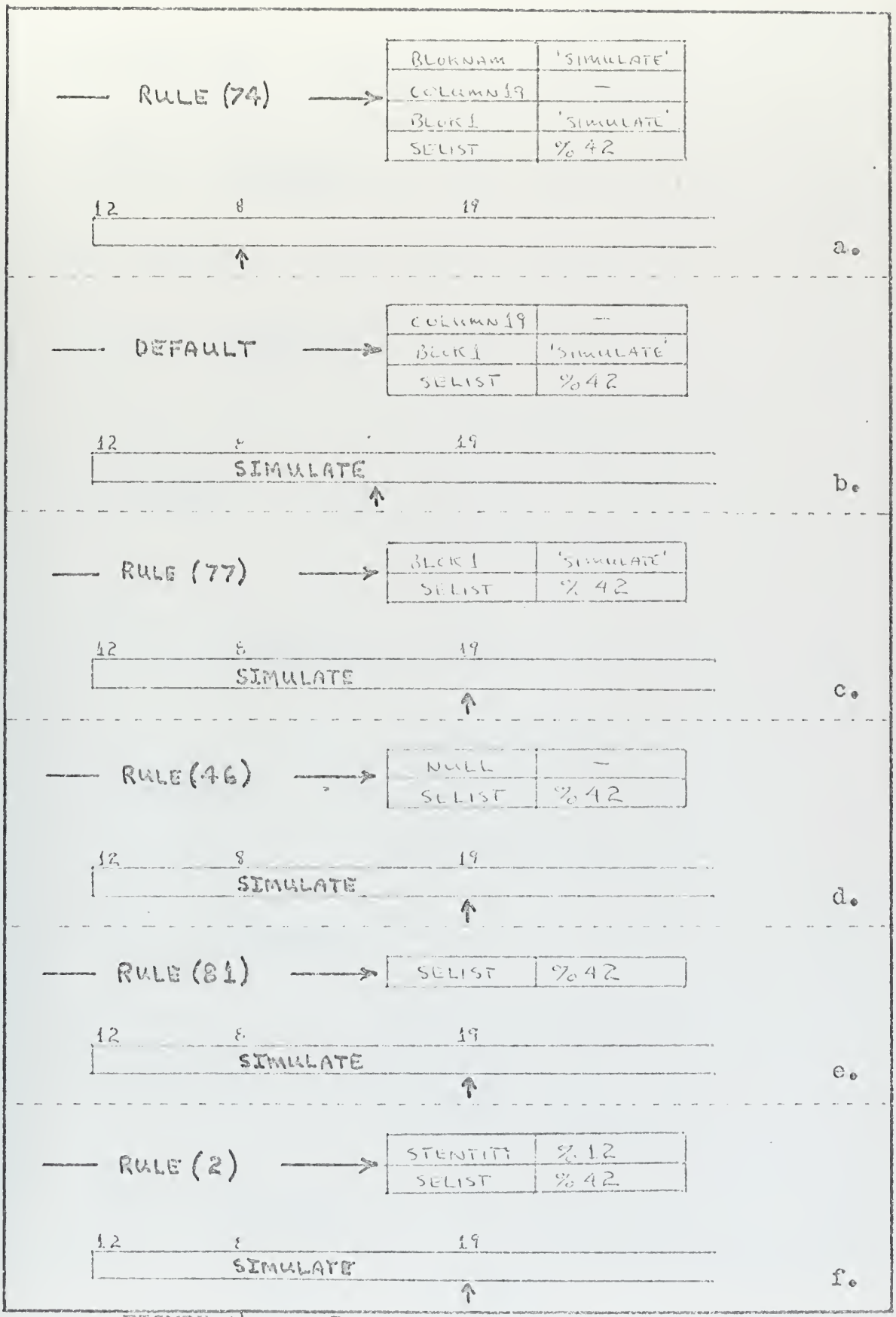


FIGURE 14 - INTERMEDIATE SAMPLE PROBLEM STEPS

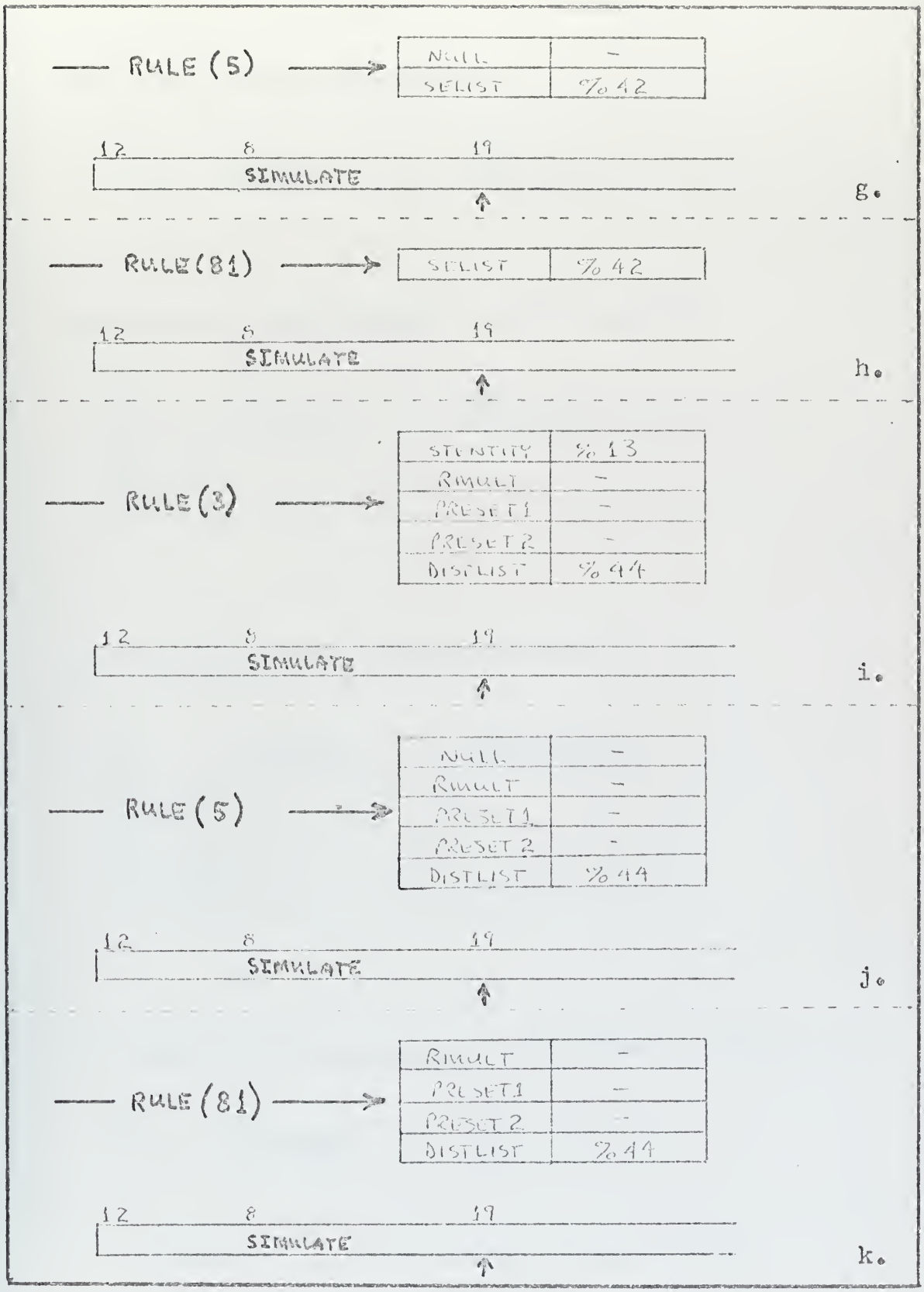


FIGURE 14 (CONTINUED)

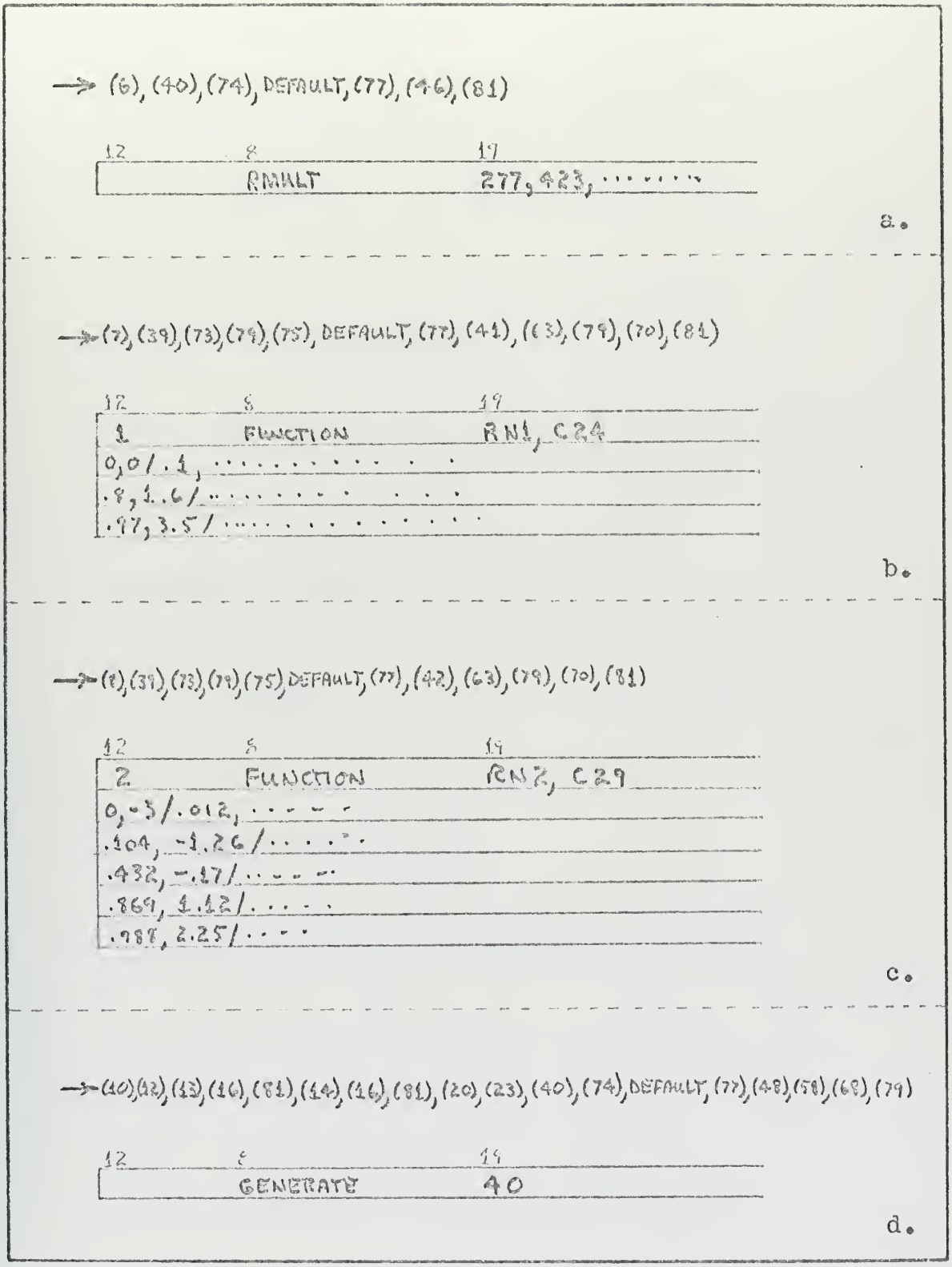


FIGURE 15 - FINAL SAMPLE PROBLEM STEPS

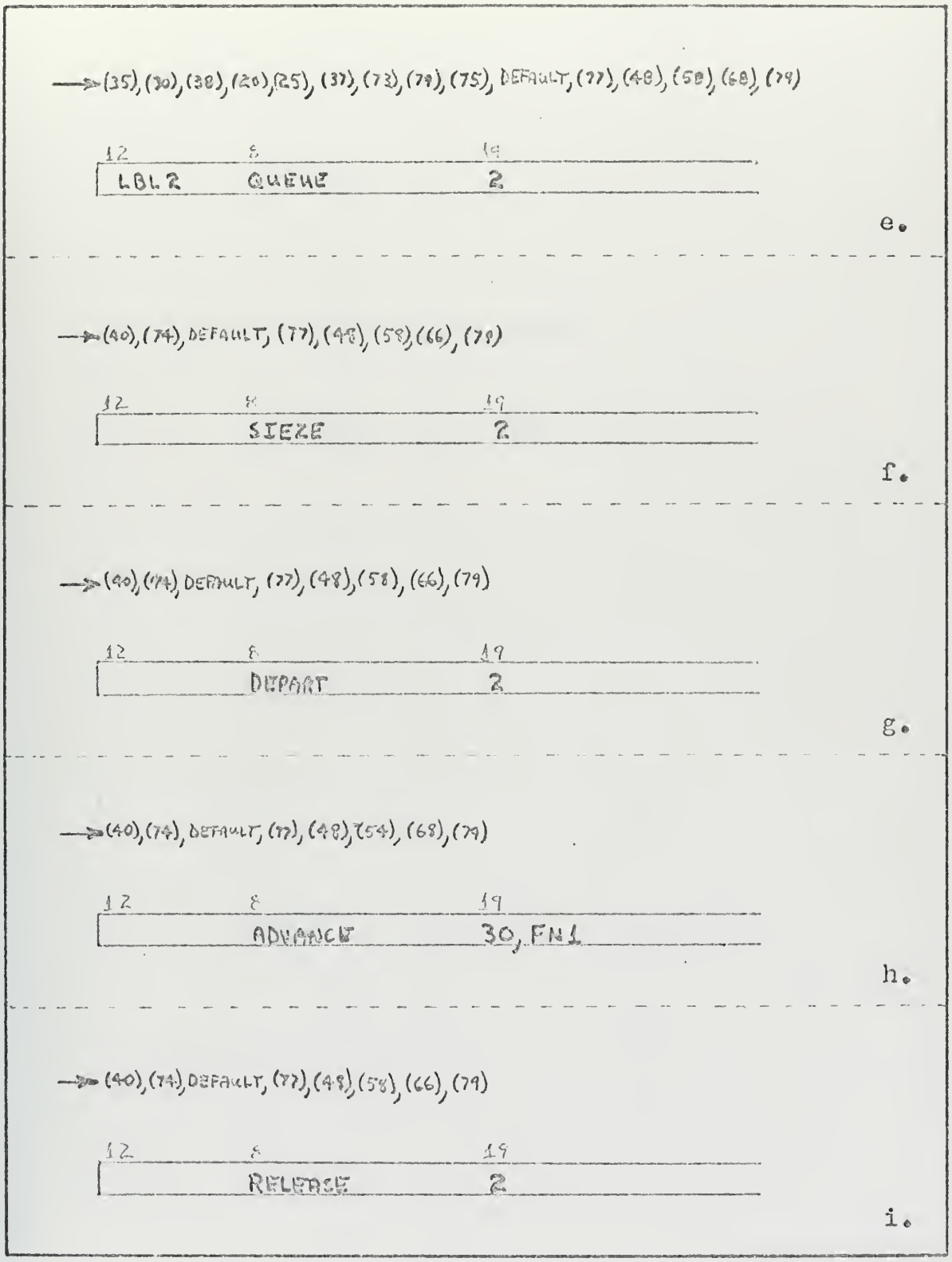


FIGURE 15 (CONTINUED)

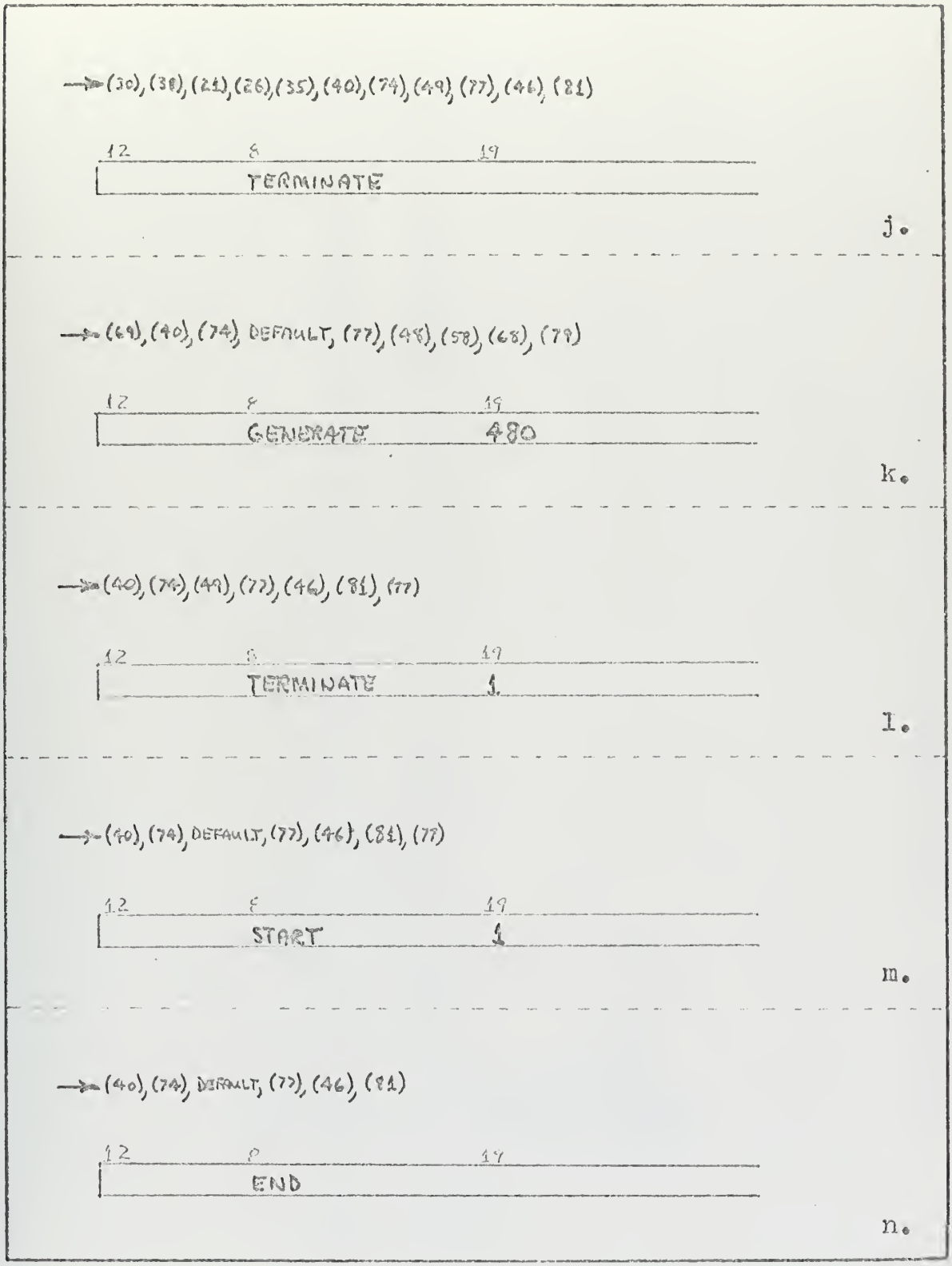


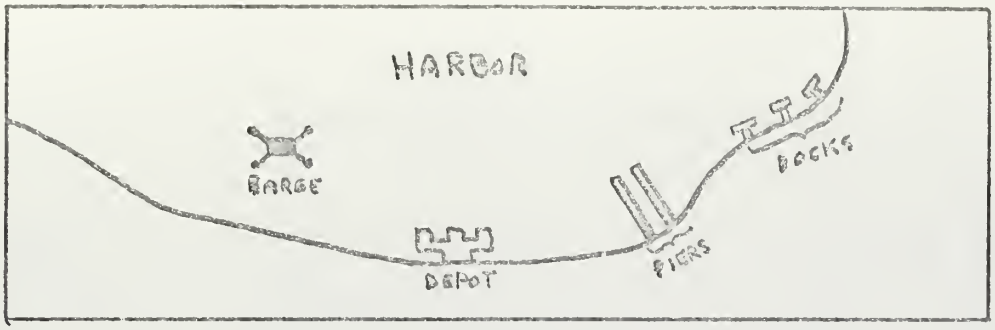
FIGURE 15 (CONTINUED)

```
        SIMULATE
        RMULT          277,423,715,121,655,531,999,813
 1      FUNCTION       RN1,C24
0,0/.1,.104/.2,.222/.3,.355/.4,.509/.5,.69/.6,.915/1.7,1.2/1.75,1.38/
.8,1.6/.84,1.83/.88,2.12/.9,2.3/.92,2.52/.94,2.81/.95,2.99/.96,3.2/
.97,3.5/.98,3.9/.99,4.6/.995,5.3/.998,6.2/.999,7/.9997,8/
 2      FUNCTION       RN2,C29
0,-3/.012,-2.25/.027,-1.93/.043,-1.72/.062,-1.54/.084,-1.38/
.104,-1.26/.131,-1.12/.159,-1/.187,-.89/.23,-.74/.267,-.62/.334,-.43/
.432,-.17/.5,0/.568,.17/.666,.43/.732,.62/.77,.74/.813,.89/.841,1/
.869,1.12/.896,1.26/.916,1.38/.938,1.54/.957,1.72/.973,1.93/
.988,2.25/1,3/
        GENERATE       40
 LBL2   QUEUE          2
        SIEZE          2
        DEPART         2
        ADVANCE        30,FN1
        RELEASE        2
        TERMINATE
        GENERATE       480
        TERMINATE      1
        START          1
        END
```

FIGURE 16 - SAMPLE PROBLEM GPSS SOLUTION CODE

IV. SYSTEM APPLICATION TO A SPECIFIC PROBLEM

Figure 17 illustrates a rudimentary port facility and provides a fact summary description of a simulation problem involving that facility. This particular problem was chosen because it allows the GES to demonstrate its capability with the use of facilities and storages, limited use of queues (i.e. single-server), time distributions, limited use of transaction parameters, multi-path transaction routing, and transaction delay based upon conditional waiting. Figure 18 shows a simplified graphic representation of the problem IDS. Figure 19 presents the problem IDS in a direct-specification format and Figure 20 shows the computer output resulting from GES processing of the problem.



There is a port containing a harbor, 3 docks, 2 piers, a depot, and a barge. Ships arrive at the port with an interarrival time of 5 hours, uniformly distributed, with a range of ± 1 hour. 50 % of the ships are blue ships, 30 % are red, and 20 % are green. After a ship arrives at the port, it unloads cargo at any available dock. Each dock has a capacity of 1 unit. Each ship takes up 1 unit of capacity. Unloading time at the dock is normally distributed as follows:

blue ship - mean of 5 hours, std dev of 1.5 hours
red ship - mean of 4 hours, std dev of 1.0 hours
green ship - mean of 3 hours, std dev of .5 hours

After unloading at a dock, a blue ship unloads cargo at the barge, a red ship unloads cargo at the depot, and a green ship unloads cargo at a pier. The barge has a capacity of 1 unit, a pier has a capacity of 1 unit, the depot has a capacity of 4 units. Unloading times are as follows:

barge - 1.5 hours, exponentially distributed
depot - 1 hour, exponentially distributed
pier - 1 hour, normally distributed, std dev of 15 minutes

Next, after these latest unloadings, 40 % of the ships load cargo at a dock, and the remainder wait in the harbor. Dock loading time is 2 hours for any ship. After loading cargo at a dock, a ship waits in the harbor. A ship waits in the harbor until the barge is unoccupied. After waiting in the harbor, a ship leaves the port. The basic time unit is the minute. Problem duration is 4 days.

FIGURE 17 - PORT FACILITY FACT SUMMARY

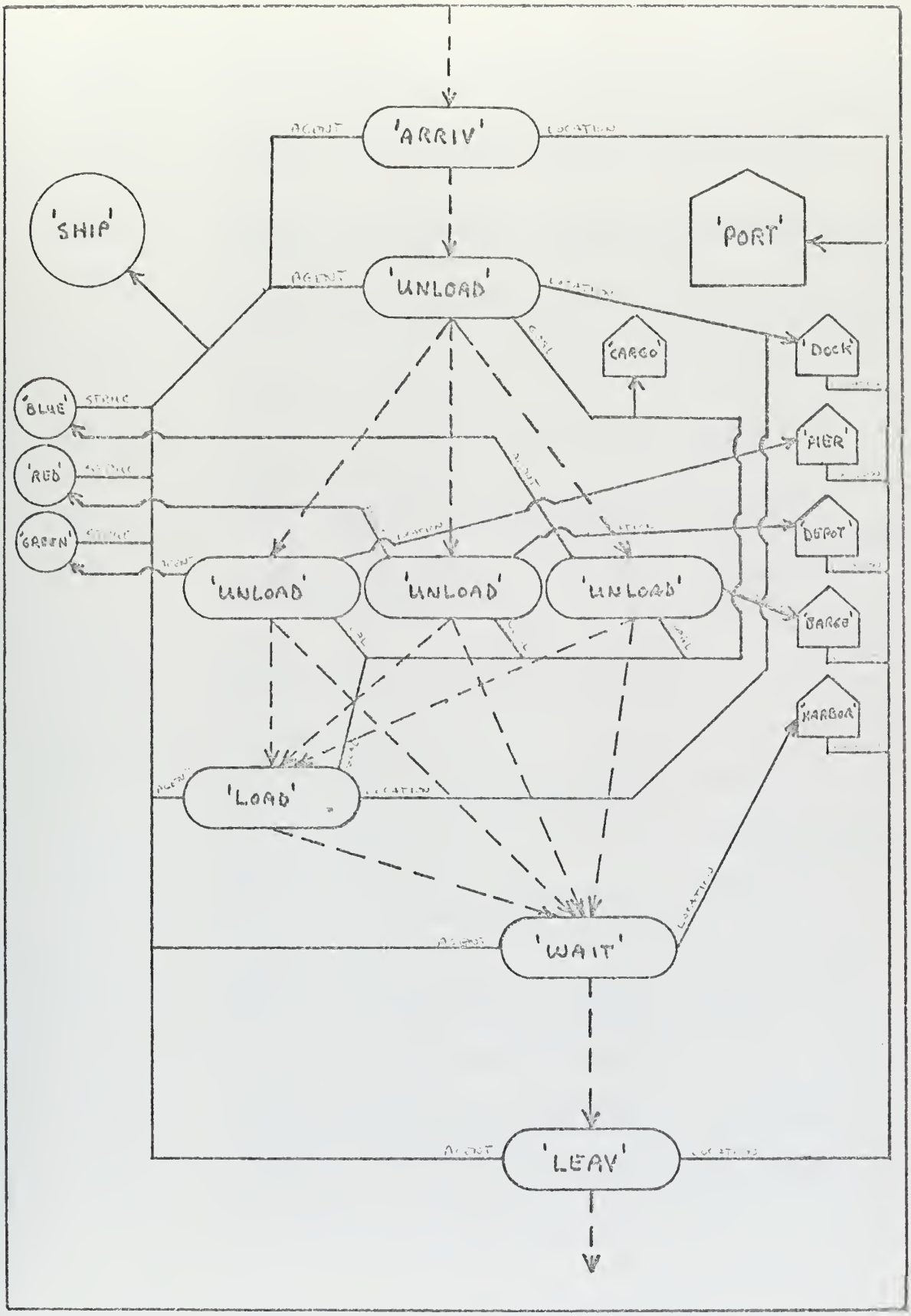


FIGURE 18 – PORT FACILITY IDS GRAPHIC REPRESENTATION

```
'REC1'    ('ARRIV',IDNO=1,SUCC='REC2',AGENT='REC11',
           LOCATION='REC71',IETM='REC41',ASNDISTR='REC47')

'REC2'    ('UNLOAD',IDNO=2,PRED='REC1',SUCC='REC61',
           AGENT='REC11',GOAL='REC13',LOCATION='REC72',
           DURATION='REC42')

'REC3'    ('UNLOAD',IDNO=3,PRED='REC2',SUCC='REC62',
           AGENT='REC15',GOAL='REC13',LOCATION='REC74',
           DURATION='REC44')

'REC4'    ('UNLOAD',IDNO=4,PRED='REC2',SUCC='REC62',
           AGENT='REC17',GOAL='REC13',LOCATION='REC75',
           DURATION='REC45')

'REC5'    ('UNLOAD',IDNO=5,PRED='REC2',SUCC='REC62',
           AGENT='REC19',GOAL='REC13',LOCATION='REC76',
           DURATION='REC46')

'REC6'    ('LOAD',IDNO=6,PRED='REC96',SUCC='REC7',
           AGENT='REC11',GOAL='REC13',LOCATION='REC72',
           DURATION='REC210')

'REC7'    ('WAIT',IDNO=7,PRED='REC97',SUCC='REC3',
           AGENT='REC11',LOCATION='REC77',DURATION='REC81')

'REC8'    ('LEAV',IDNO=8,PRED='REC7',AGENT='REC11',
           LOCATION='REC71')


'REC11'   ('SHIP',IDNO=1,CONSUMP='REC212')

'REC12'   ('PORT',IDNO=2,QUANTITY=1,STORIND=1)

'REC13'   ('CARGO',IDNO=3)

'REC14'   ('DOCK',IDNO=4,LOCATION='REC73',QUANTITY=3,
           CAPACITY='REC212',STORIND=1)

'REC15'   ('SHIP',IDNO=5,QUANTITY='REC205',COLOR='GREEN',
           STRUC='REC11')

'REC16'   ('PIER',IDNO=6,LOCATION='REC73',QUANTITY=2,
           CAPACITY='REC212',STORIND=1)

'REC17'   ('SHIP',IDNO=7,QUANTITY='REC207',COLOR='RED',
           STRUC='REC11')

'REC18'   ('DEPOT',IDNO=8,LOCATION='REC73',QUANTITY=1,
           CAPACITY='REC220',STORIND=1)

'REC19'   ('SHIP',IDNO=9,QUANTITY='REC208',COLOR='BLUE',
           STRUC='REC11')

'REC20'   ('BARGE',IDNO=10,LOCATION='REC73',QUANTITY=1,
           CAPACITY='REC212')

'REC21'   ('HARBOR',IDNO=11,LOCATION='REC73',QUANTITY=1,
           STORIND=1)
```

FIGURE 19 - PORT FACILITY IDS

```
'REC41'   ('UNIFORM',MEAN='REC201',RANGE='REC202')

'REC42'   ('NORMAL',MEAN='REC43',STDEV='REC48')

'REC43'   ('TYPTABL',FNID=3,FNARG='REC203',DORC="D",
            XYLAST=106,@101='REC15',@102='REC213',
            @103='REC17',@104='REC214',@105='REC19',
            @106='REC201')

'REC44'   ('NORMAL',MEAN='REC202',STDEV='REC206')

'REC45'   ('EXPON',MEAN='REC202')

'REC46'   ('EXPON',MEAN='REC209')

'REC47'   ('TYPDIST',FNID=4,PNUM='REC212',FNARG='REC211',
            DORC="D",XYLAST=106,@101='REC215',@102='REC15',
            @103='REC216',@104='REC17',@105='REC217',
            @106='REC19')

'REC48'   ('TYPTABL',FNID=5,FNARG='REC203',DORC="D",
            XYLAST=106,@101='REC15',@102='REC213',
            @103='REC17',@104='REC202',@105='REC19',
            @106='REC209')


'REC61'   ('PTYP',SUCARG='REC203',XYLAST=106,@101='REC15',
            @102='REC3',@103='REC17',@104='REC4',
            @105='REC19',@106='REC5')

'REC62'   ('FRACTNL',SUCARG='REC211',XYLAST=104,
            @101='REC219',@102='REC6',@103='REC217',
            @104='REC7')


'REC71'   ('AT',LOCOBJ='REC12')

'REC72'   ('AT',LOCOBJ='REC14')

'REC73'   ('IN',LOCOBJ='REC12')

'REC74'   ('AT',LOCOBJ='REC16')

'REC75'   ('AT',LOCOBJ='REC18')

'REC76'   ('AT',LOCOBJ='REC20')

'REC77'   ('IN',LOCOBJ='REC21')


'REC81'   ('COND1',CONDENTY='REC20')


'REC91'   ('MOBLIST',LASTREC=14,@11='REC11',@12='REC15',
            @13='REC17',@14='REC19')

'REC92'   ('STALIST',LASTREC=17,@11='REC12',@12='REC13',
            @13='REC14',@14='REC16',@15='REC18',
            @16='REC20',@17='REC21')
```

FIGURE 19 (CONTINUED)

```
'REC93'   ('ACTNLIST',LASTREC=18,@11='REC1',@12='REC2',
            @13='REC3',@14='REC4',@15='REC5',@16='REC6',
            @17='REC7',@18='REC8')

'REC94'   ('DSTRLIST',LASTREC=18,@11='REC41',@12='REC42',
            @13='REC43',@14='REC44',@15='REC45',@16='REC46',
            @17='REC47',@18='REC48')

'REC95'   ('SCSRLIST',LASTREC=15,@11='REC2',@12='REC61',
            @13='REC62',@14='REC7',@15='REC8')

'REC96'   ('PREDLIST',LASTREC=13,@11='REC3',@12='REC4',
            @13='REC5')

'REC97'   ('PREDLIST',LASTREC=14,@11='REC3',@12='REC4',
            @13='REC5',@14='REC6')


'REC201'  ('MINUTE',NUM=300)

'REC202'  ('MINUTE',NUM=60)

'REC203'  ('PARAMNO',NUM=1)

'REC204'  ('MINUTE',NUM=5760)

'REC205'  ('PERCENT',NUM=20)

'REC206'  ('MINUTE',NUM=15)

'REC207'  ('PERCENT',NUM=30)

'REC208'  ('PERCENT',NUM=50)

'REC209'  ('MINUTE',NUM=90)

'REC210'  ('MINUTE',NUM=120)

'REC211'  ('RANDM')

'REC212'  ('UNIT',NUM=1)

'REC213'  ('MINUTE',NUM=180)

'REC214'  ('MINUTE',NUM=240)

'REC215'  ('DECIMAL',NUM=200)

'REC216'  ('DECIMAL',NUM=500)

'REC217'  ('DECIMAL',NUM=1000)

'REC218'  ('MINUTE',NUM=30)

'REC219'  ('DECIMAL',NUM=400)

'REC220'  ('UNIT',NUM=4)


SETMEM    (RNNO(MEMORY)=1,MEPTR(MEMORY)='REC91',
            DISTPTR(MEMORY)='REC94',SUCPTR(MEMORY)='REC95',
            PRCBTIME(MEMORY)='REC204',MFNID(MEMORY)=6,
            SEPTR(MEMORY)='REC92',ACPTR(MEMORY)='REC93',
            MVARID(MEMORY)=1)
```

FIGURE 19 (CONTINUED)

```
        SIMULATE
 4      STORAGE     3
 6      STORAGE     2
 8      STORAGE     4
        RMULT       277,423,715,121,655,531,999,813
 1      FUNCTION    RN1,C24
0,0/.1,.104/.2,.222/.3,.355/.4,.509/.5,.69/.6,.915/1.7,1.2/1.75,1.38/
.8,1.6/.84,1.83/.88,2.12/.9,2.3/.92,2.52/.94,2.81/.95,2.99/.96,3.2/
.97,3.5/.98,3.9/.99,4.6/.995,5.3/.998,6.2/.999,7/.9997,8/
 2      FUNCTION    RN2,C29
0,-3/.012,-2.25/.027,-1.93/.043,-1.72/.062,-1.54/.084,-1.38/
.104,-1.26/.131,-1.12/.159,-1/.187,-.89/.23,-.74/.267,-.62/.334,-.43/
.432,-.17/.5,0/.568,.17/.666,.43/.732,.62/.77,.74/.813,.89/.841,1/
.869,1.12/.896,1.26/.916,1.38/.938,1.54/.957,1.72/.973,1.93/
.988,2.25/1,3/
 3      FUNCTION    P1,D3
5,180/7,240/9,300/
 4      FUNCTION    RN3,D3
.2,5/.5,7/1,9/
 5      FUNCTION    P1,D3
5,30/7,60/9,90/
 6      FUNCTION    P1,D3
5,LBL3/7,LBL4/9,LBL5/
 7      FUNCTION    RN4,D2
.4,LBL6/1,LBL7/
        GENERATE    300,60
        ASSIGN      1,FN4
        ENTER       2
 LBL2   QUEUE       4
        ENTER       4
        DEPART      4
        ADVANCE     V1
 1      FVARIABLE   FN3+(FN5*FN2)
        LEAVE       4
        TRANSFER    ,FN6
```

FIGURE 20 - PORT FACILITY GPSS SOLUTION CODE

```
LBL3    QUEUE       6
        ENTER       6
        DEPART      6
        ADVANCE     V2
2       FVARIABLE   60+(15*FN2)
        LEAVE       6
        TRANSFER    ,FN7
LBL4    QUEUE       8
        ENTER       8
        DEPART      8
        ADVANCE     60,FN1
        LEAVE       8
        TRANSFER    ,FN7
LBL5    QUEUE       10
        SIEZE       10
        DEPART      10
        ADVANCE     90,FN1
        RELEASE     10
        TRANSFER    ,FN7
LBL6    QUEUE       4
        ENTER       4
        DEPART      4
        ADVANCE     120
        LEAVE       4
LBL7    QUEUE       11
        ENTER       11
        DEPART      11
        GATE NU     10
        LEAVE       11
LBL8    LEAVE       2
        TERMINATE
        GENERATE    5760
        TERMINATE   1
        START       1
        END
```

FIGURE 20 (CONTINUED)

V. CONCLUSIONS AND RECOMMENDATIONS

Both the primary and secondary objectives stated in the INTRODUCTION were met. Although the GES is by no means a system to process all forms of simulation problems, significant principles have been discovered for many of the common queueing situations. Future improvements of GES will probably be limited to the expansion of present capabilities, and should not involve changes in the basic system approaches to internal data structure and conversion rules.

A. GENERAL PRINCIPLES FOR RETENTION

It is recommended that three basic principles of the GES be retained for future research in this field. They are:

(1) The primary units of IDS form are based upon the ACTION/ENTITY/ATTRIBUTE/VALUE concept.

(2) The NLP conversion rule concept and form.

(3) The ACTION-at-a-location concept as a basis for IDS form.

B. FUTURE DEVELOPMENT

Future development of the GES should include research in the following areas:

(1) System capability to process problems involving multi-server queues, shortest line choices, entity service times, complex problem durations, complex statistical requests, and multiple sub-structured entities.

(2) A printout relating entity identification numbers to problem names.

(3) A printout of the answers to the problem, not just a GPSS program.

(4) A capability to detect and request missing problem information.

APPENDIX A - BNF DESCRIPTION OF NLP ENCODING RULES

```
      <ENCODING RULE> ::= <LEFT PART> --> <RIGHT PART>

(1)   <LEFT PART> ::= <SEGMENT TYPE NAME> |
           <SEGMENT TYPE NAME>(<CONDITION SPECIFICATION>)

      <RIGHT PART> ::= <SEGMENT> | <RIGHT PART><SEGMENT>

(1)   <SEGMENT> ::= <SEGMENT TYPE NAME> |
           <SEGMENT TYPE NAME>(<LABELING SPECIFICATION>)

      <CONDITION SPECIFICATION> ::= <CONDITION ELEMENT> |
           <CONDITION SPECIFICATION><OR CONDITION> |
           <CONDITION SPECIFICATION>,<CONDITION ELEMENT>

      <LABELING SPECIFICATION> ::= <LABELING ELEMENT> |
           <LABELING SPECIFICATION>,<LABELING ELEMENT>

      <CONDITION ELEMENT> ::= <SIMPLE CONDITION ELEMENT> |
           ¬ <SIMPLE CONDITION ELEMENT> |
           <ATTRIBUTE>.<COMPARATOR>.<ATTRIBUTE>

      <OR CONDITION> ::= <OR STATEMENT> |
           <OR CONDITION><OR STATEMENT>

(2)   <OR STATEMENT> ::= <CONDITION ELEMENT> |
           <CONDITION ELEMENT>

(1)   <SIMPLE CONDITION ELEMENT> ::= <ATTRIBUTE> |
           '<NAMED RECORD NAME>' | $<CONSTANT><RECORD NAME>

(1)   <LABELING ELEMENT> ::= %<RECORD NAME> |
           -<ATTRIBUTE> | <ATTRIBUTE> |
           '<NAMED RECORD NAME>' | <ATTRIBUTE>=<EXPRESSION>

(1)   <RECORD NAME> ::= <LITERAL RECORD NAME> |
           <ATTRIBUTE NAME> |
           <ATTRIBUTE NAME>(<RECORD NAME>) |
           <ATTRIBUTE NAME>($(<RECORD NAME>))

(1)   <ATTRIBUTE> ::= <ATTRIBUTE NAME> | @<CONSTANT> |
           @<ATTRIBUTE> | <ATTRIBUTE>(<RECORD NAME>)

      <EXPRESSION> ::= <VALUE> | <VALUE><OP><CONSTANT> |
           <VALUE><OP><ATTRIBUTE>

(1)   <LITERAL RECORD NAME> ::= MEMORY | MEM | SEGMENT |
           SEG | RECORD | <SEGMENT TYPE NAME> |
           '<NAMED RECORD NAME>'

      <VALUE> ::= <ATTIBUTE> | <CONSTANT>

      <OP> ::= + | - | * | /

      <COMPARATOR> ::= EQ | NE | LT | GT | LE | GE

      <CONSTANT> ::= <DIGIT> | <CONSTANT><DIGIT>

      <DIGIT> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

NOTES: (1) SEGMENT TYPE NAME, NAMED RECORD NAME, AND ATTRIBUTE NAME ARE SELF EXPLANATORY TERMS.

(2) THE SYMBOL | IS PART OF THE <OR STATEMENT>.

APPENDIX B

NLP ENCODING RULE SYMBOLOGY AND USAGE

(1) Rules are scanned from the beginning to the end of the appropriate rule list until either an applicable rule is found or the end of the list is reached.

(2) Should a scan reach the end of a rule list without having found an applicable rule, a default procedure causes the ANMS attribute of the first SUP of the record being tested to be printed. The record is then erased.

(3) An individual rule is processed from left to right. The portion of the rule to the left of the conversion symbol (-->) is called the "condition" of the rule and is used to test for the applicability of the rule. The portion of the rule to the right of the conversion symbol is called "labeling" and is used to designate the method and extent of new record construction.

(4) A record is explicitly referenced by designating the record name (i.e. SEGMENT TYPE), or, if the record to be referenced is the one currently being tested or constructed, by designating the record name RECORD, SEGMENT, or SEG.

(5) If the record to be referenced is the one currently being tested or constructed, it may be implicitly designated by default.

(6) An attribute may be referenced either by name or by number. The number designation range of legal values is from 11 to 300 and is specified by preceeding the number with the symbol "@".

(7) MEMORY is a unique record in that no copies are made of it. All modifications of MEMORY are made on the original.

(8) Since the rules usually process copies of original records, the only way to add, delete, or change attribute information in an original record is to do so via a MEMORY designation.

(9) If a record to be constructed is to be of the same SEGMENT TYPE as the condition record, is to be a copy of the condition record, and only one such record is to be constructed, then only the SEGMENT TYPE need be listed to accomplish the construction.

(10) The encoding process is complete when no SEGMENT TYPE pointers are left on the stack.

(11) OUTPUT is a unique record in that the designation of its attributes results in printed output or printer carriage control.

e.g. @11 designated the number of lines to skip
@12 designates the column to which the printer shifts
@13 designates EBCDIC characters to be printed

@14 designates an integer to be printed
@15 designates a decimal quantity from .001 to 1.000 to be printed

(12) An OUTPUT record without designated attributes is simply erased.

(13) A single EBCDIC character will be printed if it is separated from other rule symbols by at least one blank space.

(14) --> is called the conversion symbol and means "construct the designated records which follow".

(15) (means "of", e.g. SUCC (ACT) mean "successor of action".

(16)) and , are used to assist in reading the rules; they have no significance for the rule processor.

(17) $Q means "test attribute number Q through successive records possessing attribute Q where Q is an integer and attribute Q points to a record that may possess attribute Q. Non-designation of Q will default to 1, signifying the SUP attribute.

(18) .EQ., .NE., .GT., .LT., .LE., and .GE. have the same meaning as in FORTRAN and are used for the testing of rule conditions.

(19) = has the same meaning as in FORTRAN.

(20) ¬ means "not" and is used in the testing of rule conditions.

(21) - if not used in an arithmetic expression, means "erase the designated attribute".

(22) +, -, *, and / have the standard arithmetic operator meanings. Note that they must be used with attributes of TYPE 0 cell construction and that no more than one symbol may be used in an arithmetic expression.

(23) 'XXXX' when appearing to the left of the conversion symbol, means "test to determine if the first SUP attribute of this record points to XXXX ".

(24) 'XXXX' when appearing to the right of the conversion symbol, means "assign a pointer to XXXX as the value of the first SUP attribute of this record".

(25) "ZZZ" denotes the EBCDIC representation ZZZ. These double quotes are usually used with an attribute of TYPE 1 cell construction.

(26) | means "or" and applies only to complete test conditions. Also, this symbol must be used for the rightmost test conditions as the rule condition is satisfied whenever any one of the test conditions separated by 1 is satisfied, e.g. BLOK(SUCC.EG.AGENT,GOAL.EQ.AGENT | 'TRANSFER')

(27) @Q means "attribute number Q" where Q is either an integer or an attribute of TYPE 0 cell construction.

(28) ... means "the rule is continued on the next line". This symbol is used where the rule processor could possibly have come to the end of the rule.

(29) % means "make a copy of the designated record to become the structure of this record".

(30) # means "print one blank space".

APPENDIX C

GES ATTRIBUTE LIST

The attributes used in GES are listed below according to the following format:

(#) SYMB (NAME) (TC) (POINTREC)
: DEFN

Where # is an arbitrarily assigned number for reference use only (it has no meaning in GES), SYMB is the GES name of the attribute, NAME is the English name of the attribute, TC is the attributes' cell TYPE, POINTREC is the type(s) of record(s) pointed to by the attribute (if applicable) and DEFN is a short description of the significance of the attribute.

(0) PROBTIME (problem time duration) (3) (MOBENTY/STATENTY)
: PROBTIME is the duration time of the problem.

(1) RNNO (random number) (0)
: RNNO is used as a sequential (1 to 8) random number generator identification number.

(2) TEMP (temporary value) (0)
: TEMP is used for numerical calculations and as a counter.

(3) MEPTR (mobile entity list pointer) (3) (RECLIST)
: MEPTR is used as a pointer to a list of mobile entities.

(4) SEPTR (stationary entity list pointer) (3) (RECLIST)
: SEPTR is used as a pointer to a list of stationary entities.

(5) ACPTR (action list pointer) (3) (RECLIST)
: ACPTR is used as a pointer to a list of actions.

(6) DISTPTR (distribution list pointer) (3) (RECLIST)
: DISTPTR is used as a pointer to a list of distributions.

(7) SUCPTR (successor list pointer) (3) (RECLIST)
: SUCPTR is used as a pointer to a list of successors.

(8) MFNID (master function identification) (0)
: MFNID is used as a sequential function identification number.

(9) MVARID (master variable identification) (0)
: MVARID is used as a sequential variable identification number.

(10) LISTCNTR (list counter) (0)
: LISTCNTR is used as a counter when processing a RECLIST.

(11) SUP (superset) (3) (NAMREC)
: SUP is used as the basis for record identification. It is also known as attribute 1.

(12) IDNO (identification number) (0)
: IDNO is used as an identification number for actions and entities.

(13) LOCATION (location) (3) (LOCDESCR)
: LOCATION points to a description of the location of an entity or action.

(14) PRED (predecessor) (3) (ACTIVITY/EVENT/RECLIST)
: PRED points to the action(s) which immediately preceeded the owner action.

(15) SUCC (successor) (3) (ACTIVITY/EVENT/SUCDSCR1/SUCDSCR2)
: SUCC points to the action which immediately follows the owner action or to a successor description.

(16) AGENT (agent) (3) (MOBENTY/STATENTY)
: AGENT points to the agent of an action.

(17) GOAL (goal) (3) (MOBENTY/STATENTY)
: GOAL points to the goal or object of an action.

(18) DURATION (duration) (3) (FNCTN/DISTR2/CONDESCR/QUANVAL)
: DURATION points to the duration of an action.

(19) IETM (inter event time) (3) (FNCTN/DISTR2/CONDESCR/QUANVAL)
: ITEM points to the time between successive occurrences of an action (e.g. inter arrival time).

(20) ASNDISTR (assignment distribution) (3) (FNCTN)
: ASNDISTR points to a function used to designate various types of transactions.

(21) QUANTITY (quantity) (0/3) (QUANVAL/RELTV/RELVAL)
: QUANTITY designates the quantity of an entity.

(22) SIZE (size) (3) (RELTV/RELVAL)
: SIZE points to the size of an entity.

(23) COLOR (color) (3) (NAMREC/RELTV/RELVAL)
: COLOR points to the color of an entity.

(24) WEIGHT (weight) (3) (QUANVAL/RELTV/RELVAL)
: WEIGHT points to the weight of an entity.

(25) STORIND (storage indicator) (0)
: STORIND indicates that an entity should be represented as a GPSS storage.

(26) IDENT (identity) (1) (8 PLACE EBCDIC CELL)
: IDENT is used to indicate an arbitrary entity designation (e.g. pier A and pier B).

(27) STRUC (structure) (3) (MOBENTY/STATENTY)
: STRUC points to an entity of which the owner entity is a subset.

(28) CONSUMP (consumption) (3) (QUANVAL)
: CONSUMP points to the consumption capability of a single owner entity.

(29) CAPACITY (capacity) (3) (QUANVAL)
: CAPACITY points to the capacity capability of a single owner entity.

(30) MEAN (mean) (3) (FNCTN/DISTR2/QUANVAL)
: MEAN points to the mean of a distribution.

(31) RANGE (range) (3) (FNCTN/DISTR2/QUANVAL)
: RANGE points to the range of a uniform distribution, represented as a $\pm$ number.

(32) STDEV (standard deviation) (3) (FNCTN/DISTR2/QUANVAL)
: STDEV points to the standard deviation of a normal distribution.

(33) FNID (function identification) (0)
: FNID is used as an identification number for functions.

(34) FNARG (function argument) (3) (FNCTN/ARGVAL)
: FNARG points to the value to be used as argument A in a function definition.

(35) DORC (d or c) (1) (8 PLACE EBCDIC CELL)
: DORC points to either "d" or "c" and is used to specify whether a function is discrete or continuous.

(36) XYLAST (last Y coordinate) (0)
: XYLAST designates the attribute number used to specify the last Y coordinate in a series of function definition coordinate pairs.

(37) PNUM (parameter number) (3) (QUANVAL)
: PNUM points to the parameter number to be used in a particular function definition.

(38) SUCARG (successor argument) (3) (FNCTN/ARGVAL)
: SUCARG points to the value to be used as argument A in a successor description function definition.

(39) MAXQ (maximum queue contents) (3) (QUANVAL)
: MAXQ points to the maximum queue contents allowed for certain conditional transaction routing.

(40) OPENACT (open action) (3) (ACTIVITY/EVENT)
: OPENACT points to the action to proceed to if a condition is satisfied.

(41) CLOSACT (closed action) (3) (ACTIVITY/EVENT)
: CLOSACT points to the action to proceed to if a condition is not satisfied.

(42) ARGA (argument A) (3) (FNCTN/DISTR2/ARGVAL/QUANVAL/RELTV/MOBENTY/STATENTY/SUCDSCR1/SUCDSCR2/ACTIVITY/EVENT)
: ARGA points to a record which will eventually specify argument A in a GPSS block.

(43) ARGB (argument B) (0)
: ARGB specified what may eventually become argument B in a GPSS block.

(44) LABL (label) (0)
: LABL specifies the number to be used with "LBL" in labeling related groups of GPSS blocks.

(45) BLOKMOD (block modifier) (1) (8 PLACE EBCDIC CELL)
: BLOKMOD points to a modifier of TEST or GATE blocks (e.g. "NU", "SNF", "NE", etc.)

(46) MODREL (relative modifier) (3) (RELVAL)
: MODREL points to a value which is considered to be relative (e.g. fast, slow, large, small, etc.) but is not usually specified as being relative to anything else.

(47) OBJREL (relative object) (3) (ACTIVITY/EVENT/MOBENTY/STATENTY/QUANVAL/NAMREC)
: OBJREL designates the record to which the owner record is relative.

(48) NUM (number) (0)
: NUM specifies an integer.

(49) LOCOBJ (object location) (3) (STATENTY)
: LOCOBJ points to the entity which is the basis for a location description.

(50) ANMS (attribute name) (1) (8 PLACE EBCDIC CELL)
: ANMS points to the name used when referring to a record. It is also known as attribute 12.

(51) LASTREC (last record) (0)
: LASTREC specifies the number of the attribute pointing to the last record in a RECLIST.

(52) CHARS (characters) (1) (8 PLACE EBCDIC CELL)
: CHARS is used as an intermediate storage attribute for EBCDIC characters.

(53) CONDENTY (condition entity) (3) (MOBENTY/STATENTY)
: CONDENTY points to the entity which is to be tested for a particular condition.

(54) @11-@15
: Special attributes having meaning only when used with OUTPUT segments. Usage is explained in APPENDIX B.

(55) @11-@100

: Special attributes having meaning only when used with a RECLIST record. They are used to point to the records in the list.

(56) @101-@300:

: Special attributes having meaning only when used with a FNCTN record. They are used to alternately specify the X and Y co-ordinates of a function definition.

APPENDIX D

GES RECORD TYPE LIST

The types of records used in GES are listed below according to the following format:

NAME (ATTR)
: DESCR

Where NAME is the name of the record type, ATTR is a list of normally held attributes specified by APPENDIX C reference numbers, and DESCR is a short description of what the record represents or how it is used.

MEMORY (0-10)
: MEMORY is used as a master file for identification numbers, lists of other important records, and miscellaneous data.

EVENT (2, 11-17, 19, 20)
: EVENT represents an action that has no time duration (e.g. arrive, leave, etc.)

ACTIVITY (2, 11-18, 20)
: ACTIVITY represents an action that occurs over a period of time (duration) (e.g. unload, wait, etc.)

MOBENTY (2, 11, 12, 21-24, 26-28)
: MOBENTY represents a mobile (i.e. self-propelled) entity (e.g. ship, man, etc.)

STATENTY (2, 11-13, 21-27, 29)
: STATENTY represents a stationary entity (e.g. dock, store, etc.)

FNCTN (2, 11, 30, 33-37, @101→)
: FNCTN is used to describe an attribute value that will require a function definition (e.g. empirical distribution)

DISTR2 (2, 11, 30-32)
: DISTR2 is used to describe an attribute value that refers to a pre-set program-defined function (e.g. uniform, exponential, normal)

SUCDSCR1 (2, 11, 33, 36, 38, @101→)
: SUCDSCR1 is used to represent a successor description that will require a function definition.

SUCDSCR2 (2, 11, 38-41)
: SUCDSCR2 is used to represent a successor description that depends upon the condition of a facility or storage for routing instructions.

BLOKDSCR (2, 11, 12, 35, 42-45)
: BLOKDSCR represents a GPSS block.

RECLIST (2, 11, 51, @11→)
: RECLIST represents a list of records.

NAMREC (2, 11, 50)
: NAMREC represents a named record.

LOCDESCR (2, 11, 49)
: LOCDESCR is used to describe the location of an action or entity.

RELTV (2, 11, 46, 47)
: RELTV is used to describe a relative condition of an action or entity.

SUPREC/RELVAL (2, 11)
: SUPREC/RELVAL are used when a record other than a NAMREC is required for a non-numerical value.

QUANVAL/ARGVAL (2, 11, 48)
: QUANVAL/ARGVAL are used when a numerically oriented value is required.

CONDESCR (2, 11, 53)
: CONDESCR is used to describe a conditional waiting situation. COND1 refers to waiting for a storage or facility to become available. COND2 refers to waiting for a storage or facility to become unavailable.

APPENDIX E - GES NAMED RECORD LIST

```
PROBTIME  ('ATTR')
RNNO  ('ATTR')
TEMP  ('ATTR')
MEPTR  ('ATTR')
SEPTR  ('ATTR')
ACPTR  ('ATTR')
DISTPTR  ('ATTR')
SUCPTR  ('ATTR')
MFNID  ('ATTR')
MVARID  ('ATTR')
LISTCNTR  ('ATTR')
IDNO  ('ATTR')
LOCATION  ('ATTR')
PRED  ('ATTR')
SUCC  ('ATTR')
AGENT  ('ATTR')
GOAL  ('ATTR')
DURATION  ('ATTR')
IETM  ('ATTR')
ASNDISTR  ('ATTR')
QUANTITY  ('ATTR')
SIZE  ('ATTR')
COLOR  ('ATTR')
WEIGHT  ('ATTR')
STORIND  ('ATTR')
IDENT  ('ATTR')
STRUC  ('ATTR')
CONSUMP  ('ATTR')
CAPACITY  ('ATTR')
MEAN  ('ATTR')
RANGE  ('ATTR')
STDEV  ('ATTR')
FNID  ('ATTR')
FNARG  ('ATTR')
DORC  ('ATTR')
XYLAST  ('ATTR')
PNUM  ('ATTR')
SUCARG  ('ATTR')
MAXQ  ('ATTR')
OPENACT  ('ATTR')
CLOSACT  ('ATTR')
ARGA  ('ATTR')
ARGB  ('ATTR')
LABL  ('ATTR')
BLOKMOD  ('ATTR')
MODREL  ('ATTR')
OBJREL  ('ATTR')
NUM  ('ATTR')
LOCOBJ  ('ATTR')
LASTREC  ('ATTR')
CHARS  ('ATTR')
CONDENTY  ('ATTR')


ARRIV  ('EVENT')
LEAV  ('EVENT')


WAIT  ('ACTIVITY')
UNLOAD  ('ACTIVITY')
LOAD  ('ACTIVITY')
SERVIC  ('ACTIVITY')
```

```
EVENT   ('ACTION')
ACTIVITY   ('ACTION')

SHIP   ('MOBENTY')

HARBOR   ('STATENTY')
PIER   ('STATENTY')
DOCK   ('STATENTY')
PORT   ('STATENTY')
DEPOT   ('STATENTY')
BARGE   ('STATENTY')
CARGO   ('STATENTY')

MOBENTY   ('ENTITY')
STATENTY   ('ENTITY')

MOBLIST   ('RECLIST')
STALIST   ('RECLIST')
DSTRLIST   ('RECLIST')
SCSRLIST   ('RECLIST')
ACTNLIST   ('RECLIST')
PREDLIST   ('RECLIST')

EMPDIST   ('DISTR1')
TYPDIST   ('DISTR1')

UNIFORM   ('DISTR2')
EXPON   ('DISTR2')
NORMAL   ('DISTR2')

TYPTABL   ('TABL')

DISTR1   ('FNCTN')
TABL   ('FNCTN')

COND1   ('COND')
COND2   ('COND')

IN   ('LOCDESCR')
ON   ('LOCDESCR')
NEAR   ('LOCDESCR')
AT   ('LOCDESCR')
AROUND   ('LOCDESCR')

FRACTNL   ('SUCDSCR1')
PTYP   ('SUCDSCR1')

QTYP   ('SUCDSCR2')
STYP   ('SUCDSCR2')
FTYP   ('SUCDSCR2')

SUCDSCR1   ('SUCDESCR')
SUCDSCR2   ('SUCDESCR')
```

```
GT   ('RELIND1')
NG   ('RELIND1')
EQ   ('RELIND1')
NE   ('RELIND1')
LT   ('RELIND1')
NL   ('RELIND1')

ER   ('RELIND2')
EST  ('RELIND2')

RELIND1  ('RELTV')
RELIND2  ('RELTV')

FAST  ('RELTIME')
SLOW  ('RELTIME')

MANY  ('RELQNTY')
FEW   ('RELQNTY')

DARK   ('RELCOLR')
LIGHT1  ('RELCOLR')

HEAVY  ('RELWEIT')
LIGHT2  ('RELWEIT')

LARGE  ('RELSIZ')
SMALL  ('RELSIZ')

RELTIME  ('RELVAL')
RELQNTY  ('RELVAL')
RELCOLR  ('RELVAL')
RELWEIT  ('RELVAL')
RELSIZ   ('RELVAL')

HOUR   ('ABSTIME')
MINUTE  ('ABSTIME')
SECOND  ('ABSTIME')

TON   ('ABSWEIT')
POUND  ('ABSWEIT')
OUNCE  ('ABSWEIT')

DECIMAL  ('ABSQNTY')
UNIT  ('ABSQNTY')
PERCENT  ('ABSQNTY')

RED   ('ABSCOLR')
ORANGE  ('ABSCOLR')
YELLOW  ('ABSCOLR')
GREEN  ('ABSCOLR')
BLUE  ('ABSCOLR')
VIOLET  ('ABSCOLR')
BLACK  ('ABSCOLR')
WHITE  ('ABSCOLR')
```

```
ABSTIME   ('QUANVAL')
ABSWEIT   ('QUANVAL')
ABSQNTY   ('QUANVAL')

ABSCOLR   ('QUALVAL')

PARAMNO   ('ARGVAL')
FUNCNO    ('ARGVAL')
RANDM     ('ARGVAL')

RELVAL    ('VAL')
QUANVAL   ('VAL')
QUALVAL   ('VAL')
ARGVAL    ('VAL')
```

APPENDIX F

GES SEGMENT TYPE LIST

The types of segments used in GES are listed below according to the following format:

NAME (RT) (ATTR)
: DEFN

Where NAME is the name of the segment type, RT is the record type normally associated with the segment type, ATTR is a list of attributes normally associated with the segment type if RT is not an adequate description, and DEFN is a short description of the use of the segment type.

GPSSPROG
: GPSSPROG is used to initiate the GES encoding process. It has no attributes.

SELIST (RECLIST)
: SELIST is used to represent the list of all stationary entities in the problem IDS.

DISTLIST (RECLIST)
: DISTLIST is used to represent the list of all distributions in the problem IDS.

SUCLIST (RECLIST)
: SUCLIST is used to represent the list of all successors in the problem IDS.

ACLIST (RECLIST)
: ACLIST is used to represent the list of all actions in the problem IDS.

STENTITY (STATENTY)
: STENTITY is used to represent a specific stationary entity so that it may be checked for a possible storage definition printout.

FNDEF (FNCTN/DISTR2)
: FNDEF is used to represent the information necessary for a possible function definition printout.

ACT (ACTIVITY/EVENT)
: ACT is used to represent an action.

SUCREC (SUCDSCR1/SUCDSCR2/ACTIVITY/EVENT)
: SUCREC is used to represent a successor or successor description so that it may be checked for a possible function definition printout.

FNDEF1/FNDEF2 (FNCTN/DISTR2/SUCDSCR1/SUCDSCR2)
: FNDEF1/FNDEF2 are used to convert the function definition X-Y coordinate points into the printout of function definition follower blocks.

BLOK (BLOKDSCR)
: BLOK is used to represent a GPSS block.

SUCCBLOK (BLOKDSCR)
: SUCCBLOK is used to represent an intermediate step between ACT processing and BLOK processing for successor segments. This intermediate step allows the conversion of special successors to appropriate forms.

CONDBLOK (CONDESCR)
: CONDBLOK is the intermediate step used to change an ADVANCE block into a GATE blok when conditional waiting is encountered.

BLOK1 (BLOKDSCR)
: BLOK1 is used to initiate processing of the argument portion of a GPSS block.

BLOKNAM (SUPREC)
: BLOKNAM is used to print out the name of a GPSS block.

ARGS/ARG (SUCDSCR1/SUCDSCR2/ACTIVITY/EVENT/MOBENTY/STATENTY/FNCTN/DISTR2/ QUANVAL/ARGVAL)
: ARGS/ARG are used for the final processing of the argument portion of a GPSS block.

NAME [CHARS]
: NAME is used to represent EBCDIC characters.

NUMBER [NUM]
: NUMBER is used to represent an integer.

DECNUMB [NUM]
: DECNUMB is used to represent a decimal number.

OUTPUT [@11-@15]
: OUTPUT usage is explained in APPENDIX B.

RMULT
: RMULT causes RMULT information to be eventually printed.

PRESET1
: PRESET1 causes a preset exponential function to be defined as FN1 and printed.

PRESET2
: PRESET2 causes a preset normal function to be defined as FN2 and printed.

NULL
: NULL has no effect on other records or on printout. It is a dead end.

FINIS
: FINIS causes the problem duration block to be printed.

RNCHK
: RNCHK sequences the RNNO attribute in MEMORY from 1 to 8.

NEWLINE1
: NEWLINE1 shifts the printer carriage control to a new line, column 1.

NEWLINE2
: NEWLINE2 shifts the printer carriage control to a new line, column 2.

NEWLINE8
: NEWLINE8 shifts the printer carriage control to a new line, column 8.

COLUMN8
: COLUMN8 shifts the printer carriage control to column 8.

COLUMN13
: COLUMN13 shifts the printer carriage control to column 13.

COLUMN19
: COLUMN19 shifts the printer carriage control to column 19.

APPENDIX G - GES ENCODING RULES

```
(1)   GPSSPROG   -->   BLOK('SIMULATE') ...
                      SELIST(%SEPTR(MEMORY),LISTCNTR(MEMORY)=11)

(2)   SELIST(LISTCNTR(MEMORY).LT.LASTREC)   --> ...
                      STENTITY(%@LISTCNTR(MEMORY)(SELIST)) ...
                      SELIST(LISTCNTR(MEMORY)=LISTCNTR(MEMORY)+1)

(3)   SELIST   -->   STENTITY(%@LISTCNTR(MEMORY)(SELIST)) ...
                      RMULT   PRESET1   PRESET2 ...
                      DISTLIST(%DISTPTR(MEMORY),LISTCNTR(MEMORY)=11)

(4)   STENTITY(STORIND,CAPACITY)   -->   BLOK('STORAGE',IDNO=IDNO
                      (STENTITY),ARGB=QUANTITY(STENTITY)*NUM(CAPACITY(STENTITY)))

(5)   STENTITY   -->   NULL

(6)   RMULT   -->   BLOK('RMULT') ...
                      OUTPUT(@13="277,423,715,121,655,531,999,813")

(7)   PRESET1   -->   BLOK('FUNCTION',IDNO=1) ...
                      OUTPUT(@11=1,@12=1,@13="0.0/.1,.104/.2,.222/.3,.355/") ...
                      OUTPUT(@13=".4,.509/.5,.69/.6,.915/1.7,1.2/1.75,1.38/") ...
                      OUTPUT(@11=1,@12=1,@13=".8,1.6/.84,1.83/.88,2.12/.9,") ...
                      OUTPUT(@13="2.3/.92,2.52/.94,2.81/.95,2.99/.96,3.2/") ...
                      OUTPUT(@11=1,@12=1,@13=".97,3.5/.98,3.9/.99,4.6/.995") ...
                      OUTPUT(@13=",5.3/.998,6.2/.999,7/.9997,8/")
```

```
(8)   PRESET2   -->   BLOK('FUNCTION',IDNO=2) ...
                     OUTPUT(@11=1,@12=1,@13="0,-3/.012,-2.25/.027,-1.93/") ...
                     OUTPUT(@13=".043,-1.72/.062,-1.54/.084,-1.38/") ...
                     OUTPUT(@11=1,@12=1,@13=".104,-1.26/.131,-1.12/.159,") ...
                     OUTPUT(@13="-1/.187,-.89/.23,-.74/.267,-.62/.334,-.43/") ...
                     OUTPUT(@11=1,@12=1,@13=".432,-.17/.5.0/.568,.17/.666,") ...
                     OUTPUT(@13=".43/.732,.62/.77,.74/.813,.89/.841,1/") ...
                     OUTPUT(@11=1,@12=1,@13=".869,1.12/.896,1.26/.916,1.38/") ...
                     OUTPUT(@13=".938,1.54/.957,1.72/.973,1.93/") ...
                     OUTPUT(@11=1,@12=1,@13=".988,2.25/1,3/")


(9)   DISTLIST(LISTCNTR(MEMORY).LT.LASTREC)   --> ...
                     FNDEF(%@LISTCNTR(MEMORY)(DISTLIST)) ...
                     DISTLIST(LISTCNTR(MEMORY)=LISTCNTR(MEMORY)+1)

(10)  DISTLIST   -->   FNDEF(%@LISTCNTR(MEMORY)(DISTLIST)) ...
                     SUCLIST(%SUCPTR(MEMORY),LISTCNTR(MEMORY)=11)


(11)  FNDEF($'DISTR1'|$'TABL')   -->   BLOK('FUNCTION',IDNO=FNID(FNDEF),
                     ARGA=FNARG(FNDEF),TEMP=XYLAST(FNDEF)-100,
                     ARGB=TEMP/2,DORC=DORC(FNDEF),TEMP(MEMORY)=101) ...
                     NEWLINE1   FNDEF1(%FNDEF)

(12)  FNDEF   -->   NULL


(13)  SUCLIST(LISTCNTR(MEMORY).LT.LASTREC)   --> ...
                     SUCREC(%@LISTCNTR(MEMORY)(SUCLIST),
                     FNID(@LISTCNTR(MEMORY)(SUCLIST))=MFNID(MEMORY)) ...
                     SUCLIST(LISTCNTR(MEMORY)=LISTCNTR(MEMORY)+1)

(14)  SUCLIST   -->   SUCREC(%@LISTCNTR(MEMORY)(SUCLIST),
                     FNID(@LISTCNTR(MEMORY)(SUCLIST))=MFNID(MEMORY)) ...
                     ACLIST(%ACPTR(MEMORY),LISTCNTR(MEMORY)=11)


(15)  SUCREC($'SUCDSCR1')   -->   BLOK('FUNCTION',IDNO=MFNID(MEMORY),
                     MFNID(MEMORY)=MFNID(MEMORY)+1,ARGA=SUCARG(SUCREC),
                     TEMP=XYLAST(SUCREC)-100,ARGB=TEMP/2,DORC="D",
                     TEMP(MEMORY)=101)   NEWLINE1   FNDEF1(%SUCREC)
```

```
(16)   SUCREC    -->    NULL

(17)   FNDEF1    -->    ARG(%@TEMP(MEMORY)(FNDEF1),TEMP(MEMORY)=TEMP
                        (MEMORY)+1)    ,    ARG(%@TEMP(MEMORY)(FNDEF1)) ...
                        /    FNDEF2(%FNDEF1)


(18)   FNDEF2(TEMP(MEMORY),NE,XYLAST)    --> ...
                        FNDEF1(%FNDEF2,TEMP(MEMORY)=TEMP(MEMORY)+1)

(19)   FNDEF2    -->    NULL


(20)   ACLIST(LISTCNTR(MEMORY),LT,LASTREC)    --> ...
                        ACT(%@LISTCNTR(MEMORY)(ACLIST)) ...
                        ACLIST(LISTCNTR(MEMORY)=LISTCNTR(MEMORY)+1)

(21)   ACLIST    -->    ACT(%@LISTCNTR(MEMORY)(ACLIST),
                        LISTCNTR(MEMORY)=LISTCNTR(MEMORY)+1)    FINIS


(22)   ACT('ARRIV',¬PRED,ASNDISTR)    --> ...
                        BLOK('GENERATE',ARGA=IETM(ACT)) ...
                        BLOK('ASSIGN',ARGA=ASNDISTR(ACT)) ...
                        BLOK('ENTER',ARGA=LOCOBJ(LOCATION(ACT))) ...
                        SUCCBLOK(ARGA=SUCC(ACT))

(23)   ACT('ARRIV',¬PRED)    -->    BLOK('GENERATE',ARGA=IETM(ACT)) ...
                        BLOK('ENTER',ARGA=LOCOBJ(LOCATION(ACT))) ...
                        SUCCBLOK(ARGA=SUCC(ACT))

(24)   ACT(£'ACTIVITY',STORIND(LOCOBJ(LOCATION)))    --> ...
                        BLOK('QUEUE',LABL=IDNO(ACT),ARGA=LOCOBJ(LOCATION(ACT))) ...
                        BLOK('ENTER',ARGA=LOCOBJ(LOCATION(ACT))) ...
                        BLOK('DEPART',ARGA=LOCOBJ(LOCATION(ACT))) ...
                        BLOK('ADVANCE',ARGA=DURATION(ACT)) ...
                        BLOK('LEAVE',ARGA=LOCOBJ(LOCATION(ACT))) ...
                        SUCCBLOK(ARGA=SUCC(ACT))
```

```
(25)   ACT($'ACTIVITY')   -->   BLOK('QUEUE',LABL=IDNO(ACT),
                                 ARGA=LOCOBJ(LOCATION(ACT))) ...
                                 BLOK('SIEZE',ARGA=LOCOBJ(LOCATION(ACT))) ...
                                 BLOK('DEPART',ARGA=LOCOBJ(LOCATION(ACT))) ...
                                 BLOK('ADVANCE',ARGA=DURATION(ACT)) ...
                                 BLOK('RELEASE',ARGA=LOCOBJ(LOCATION(ACT))) ...
                                 SUCCBLOK(ARGA=SUCC(ACT))

(26)   ACT('LEAV',¬SUCC)   -->   BLOK('LEAVE',LABL=IDNO(ACT),
                                 ARGA=LOCOBJ(LOCATION(ACT)))   BLOK('TERMINAT')


(27)   SUCCBLOK(ARGA$'QTYP')   --> ...
                                 BLOK('TEST',BLOKMOD="L",ARGA(SUCCBLOK)) ...
                                 BLOK('TRANSFER',ARGA=OPENACT(ARGA(SUCCBLOK)))

(28)   SUCCBLOK(ARGA$'FTYP')   --> ...
                                 BLOK('GATE',BLOKMOD="NU",ARGA(SUCCBLOK)) ...
                                 BLOK('TRANSFER',ARGA=OPENACT(ARGA(SUCCBLOK)))

(29)   SUCCBLOK(ARGA$'STYP')   --> ...
                                 BLOK('GATE',BLOKMOD="SNF",ARGA(SUCCBLOK)) ...
                                 BLOK('TRANSFER',ARGA=OPENACT(ARGA(SUCCBLOK)))

(30)   SUCCBLOK   -->   BLOK('TRANSFER',ARGA(SUCCBLOK))


(31)   CONDBLOK('COND1',STORIND(CONDENTY))   --> ...
                                 BLOK('GATE',BLOKMOD="SNF",ARGA=CONDENTY(CONDBLOK))

(32)   CONDBLOK('COND1')   --> ...
                                 BLOK('GATE',BLOKMOD="NU",ARGA=CONDENTY(CONDBLOK))

(33)   CONDBLOK('COND2',STORIND(CONDENTY))   --> ...
                                 BLOK('GATE',BLOKMOD="SF",ARGA=CONDENTY(CONDBLOK))

(34)   CONDBLOK('COND2')   --> ...
                                 BLOK('GATE',BLOKMOD="U",ARGA=CONDENTY(CONDBLOK))


(35)   BLOK(¬STORIND(ARGA),'ENTER'|'LEAVE')   -->   NULL

(36)   BLOK('ADVANCE',ARGA$'COND')   -->   CONDBLOK(%ARGA(BLOK))
```

```
(37)  BLOK(LABL)   -->  NEWLINE2   L  B  L   NUMBER(NUM=LABL(BLOK)) ...
                COLUMN8   BLOKNAM(SUP(BLOK))   COLUMN19   BLOK1(%BLOK)

(38)  BLOK('TRANSFER',ARGA.EQ.@LISTCNTR(MEMORY)
                (ACPTR(MEMORY)))   -->   NULL

(39)  BLOK(IDNO)   -->  NEWLINE2   NUMBER(NUM=IDNO(BLOK)) ...
                COLUMN8   BLOKNAM(SUP(BLOK))   COLUMN19   BLOK1(%BLOK)

(40)  BLOK   -->   NEWLINE8   BLOKNAM(SUP(BLOK))   COLUMN19   BLOK1(%BLOK)


(41)  BLOK1('FUNCTION',IDNO.EQ.1)   --> ...
                ARG('RANDM')   ,  C  2  4

(42)  BLOK1('FUNCTION',IDNO.EQ.2)   --> ...
                ARG('RANDM')   ,  C  2  9

(43)  BLOK1('FUNCTION')   -->   ARG(%ARGA(BLOK1)) ...
                ,   NAME(CHARS=DORC(BLOK1))   NUMBER(NUM=ARGB(BLOK1))

(44)  BLOK1('STORAGE')   -->   NUMBER(NUM=ARGB(BLOK1))

(45)  BLOK1('TRANSFER')   -->   ,   ARG(%ARGA(BLOK1))

(46)  BLOK1(¬ARGA)   -->   NULL

(47)  BLOK1(BLOKMOD)   -->   COLUMN13   NAME(CHARS=BLOKMOD(BLOK1)) ...
                COLUMN19   ARGS(%ARGA(BLOK1))

(48)  BLOK1   -->   ARGS(%ARGA(BLOK1))


(49)  BLOKNAM('TERMINAT')   -->   T E R M I N A T E

(50)  BLOKNAM('FVARIABL')   -->   F V A R I A B L E


(51)  ARGS('EMPDIST')   -->   ARG(%MEAN(ARGS))   ,   ARG(%ARGS)

(52)  ARGS('TYPDIST')   -->   ARG(%PNUM(ARGS))   ,   ARG(%ARGS)

(53)  ARGS('UNIFORM')   -->   ARG(%MEAN(ARGS))   ,   ARG(%RANGE(ARGS))
```

```
(54)   ARGS('EXPON')   -->   ARG(%MEAN(ARGS))   ,  F  N  1

(55)   ARGS('NORMAL')   -->   V   NUMBER(NUM=MVARID(MEMORY)) ...
           BLOK('FVARIABL',IDNO=MVARID(MEMORY),MVARID(MEMORY)=
           MVARID(MEMORY)+1)   ARG(%MEAN(ARGS))   +  ( ...
           ARG(%STDEV(ARGS))   *  F  N  2  )

(56)   ARGS('QTYP')   -->   Q   ARG(%SUCARG(ARGS))   , ...
           ARG(%MAXQ(ARGS))   ,   ARG(%CLOSACT(ARGS))

(57)   ARGS('STYP'|'FTYP')   -->   ARG(%SUCARG(ARGS))   , ...
           ARG(%CLOSACT(ARGS))

(58)   ARGS   -->   ARG(%ARGS)


(59)   ARG($'FNCTN'|$'SUCDSCR1')   -->   F  N   NUMBER(NUM=FNID(ARG))

(60)   ARG('EXPON')   -->   F  N  1

(61)   ARG('NORMAL')   -->   F  N  2

(62)   ARG('PARAMNO')   -->   P   NUMBER(NUM(ARG))

(63)   ARG('RANDM')   -->   R  N   NUMBER(NUM=RNNO(MEMORY))   RNCHK

(64)   ARG('FUNCNO')   -->   F  N   NUMBER(NUM(ARG))

(65)   ARG($'ACTION')   -->   L  B  L   NUMBER(NUM=IDNO(ARG))

(66)   ARG($'ENTITY')   -->   NUMBER(NUM=IDNO(ARG))

(67)   ARG('DECIMAL')   -->   DECNUMB(NUM(ARG))

(68)   ARG($'QUANVAL')   -->   NUMBER(NUM(ARG))


(69)   FINIS   -->   BLOK('GENERATE',ARGA=PROBTIME(MEMORY)) ...
           BLOK('TERMINAT')   COLUMN19   1   BLOK('START') ...
           COLUMN19   1   BLOK('END')
```

```
(70)   RNCHK(RNNO(MEMORY).LT.8)    --> ...
                  NULL(RNNO(MEMORY)=RNNO(MEMORY)+1)
(71)   RNCHK    -->    NULL(RNNO(MEMORY)=1)

(72)   NEWLINE1    -->    OUTPUT(@11=1,@12=1)
(73)   NEWLINE2    -->    OUTPUT(@11=1,@12=2)
(74)   NEWLINE8    -->    OUTPUT(@11=1,@12=8)

(75)   COLUMN8    -->    OUTPUT(@12=8)
(76)   COLUMN13    -->    OUTPUT(@12=13)
(77)   COLUMN19    -->    OUTPUT(@12=19)

(78)   NAME    -->    OUTPUT(@13=CHARS(NAME))

(79)   NUMBER    -->    OUTPUT(@14=NUM(NUMBER))

(80)   DECNUMB    -->    OUTPUT(@15=NUM(NUMBER))

(81)   NULL    -->    OUTPUT
```

LIST OF REFERENCES

1. Chomsky, Noam, Aspects of the Theory of Syntax, MIT Press, Cambridge, Mass., 1965.

2. Lamb, Sydney M., Outline of Stratificational Grammar, Revised edition, Georgetown University Press, Washington, D. C., 1966.

3. Lamb, Sydney M., "Linguistic and Cognitive Networks", to appear in Cognition: A Multiple View (Garvin, ed.), in press.

4. Simmons, Robert F., "Automated Language Processing", in Annual Review of Information Science and Technology (Cuadra, ed.), Volume 1, Interscience Publishers, New York, N. Y., 1966.

5. Bobrow, D. G., Fraser, J. B., and Quillian, M. R., "Automated Language Processing", in Annual Review of Information Science and Technology (Cuadra, ed.), Volume 2, Interscience Publishers, New York, N. Y., 1967.

6. Salton, Gerard, "Automated Language Processing", in Annual Review of Information Science and Technology (Cuadra, ed.), Volume 3, Encyclopedia Britannica, Inc., Chicago, Ill., 1968.

7. Minsky, Marvin (ed.) Semantic Information Processing, MIT Press, Cambridge, Mass., 1968.

8. Simmons, Robert F., "Natural Language Question-Answering Systems", Comm. ACM 13, (January 1970), 15-30.

9. International Business Machines Corporation, General Purpose Simulation System/360, Introductory User's Manual, 1967.

10. International Business Machines Corporation, General Purpose Simulation System/360, User's Manual, 1968.

11. The RAND Corporation, Programming by Questionnaire: How to Construct a Program Generator, MEMORANDUM RM-5129-PR, November 1966.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	Assistant Professor G. E. Heidorn, Code 55Hd Department of Operations Analysis Naval Postgraduate School Monterey, California 93940	5
4.	LCDR Richard Carl Hansen, USN P. O. Box 993 Carmel, California 93921	1
5.	LTJG Robert C. Bolles, USNR Instructor, Code 53Bq Department of Mathematics Naval Postgraduate School Monterey, California 93940	1

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1 ORIGINATING ACTIVITY (Corporate author)	2a. REPORT SECURITY CLASSIFICATION
Naval Postgraduate School Monterey, California 93940	Unclassified
	2b. GROUP

3 REPORT TITLE

G E S : A Data-Structure-to-GPSS Encoding System

4 DESCRIPTIVE NOTES (Type of report and, inclusive dates)

Master's Thesis; December 1970

5 AUTHOR(S) (First name, middle initial, last name)

Richard Carl Hansen

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
December 1970	76	11

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
b. PROJECT NO	
c.	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)
d.	

10. DISTRIBUTION STATEMENT

"This document has been approved for public release and sale; its distribution is unlimited".

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	Naval Postgraduate School Monterey, California 93940

13. ABSTRACT

A research effort currently underway at the Naval Postgraduate School deals with the design and implementation of a computer system for translating natural language descriptions of simulation problems into executable computer programs. In this system, English text is translated into an internal data structure, which can be considered to be the computer's "mental image" of a simulation problem. This internal data structure is then translated into a computer program which will perform the simulation.

This thesis reports on the design of an appropriate internal data structure for conveniently representing simple queueing problems and on the development of a procedure for translating such a data structure into a GPSS program. Also, pertinent aspects of the overall system are briefly described, and an example application to a specific problem is included.

DD FORM 1 NOV 65 1473 (PAGE 1)

S/N 0101-807-6811

Security Classification

A-31408

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Natural language processor						
Automated GPSS